VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce přepravních nádob pomocí laserového skeneru Sick Lms 400

Detection of Shipping Containers Using the Sick Lms 400 Laser Scanner

2018 Bc. Marek Duda

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Marek Duda**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce přepravních nádob pomocí laserového skeneru Sick Lms 400
Detection of Shipping Containers Using the Sick Lms 400 Laser Scanner

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem této práce bude vytvoření Windows služby, která bude zajišťovat komunikaci s laserovým skenerem Sick Lms 400, a z naskenovaných dat provede detekci přepravních nádob. Informace o jejich poloze předá PLC prostřednictvím standardizované technologie OPC. Součástí práce bude také aplikace, která umožní zobrazit naskenované data pomocí 3D modelu.

1. Vypracování návrhu řešení.
2. Vytvoření ovladače pro komunikaci s laserovým skenerem Sick Lms 400.
3. Vytvoření Windows služby pro zpracování naskenovaných dat.
4. Tvorba algoritmu pro efektivní nalezení bodů přepravek z naměřených dat.
5. Tvorba algoritmu pro vypočítání souřadnic pro jednotlivé přepravní nádoby.
6. Vytvoření komunikace s OPC serverem.
7. Vytvoření 3D modelu pro zobrazení naměřených dat.
8. Testování (Unit testy).
9. Tvorba dokumentace.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jiří Zawada**

Konzultant diplomové práce: Ing. Jan Kožusznik, Ph.D.

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

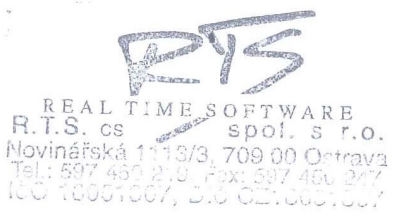
Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018

.....



Rád bych na tomto místě poděkoval firmě R.T.S. cs, spol. s r.o. za umožnění vypracovat tuto diplomovou práci a zejména Ing. Jiřímu Zawadovi za poskytnutí pomoci a příkladnému vedení během vykonávání této diplomové práce. Dále bych chtěl poděkovat panu Ing. Janu Kožusznikovi, Ph.D. za vedení během vytváření této diplomové práce v rámci školy.

Abstrakt

Tato diplomová práce se zabývá vyřešením problematiky vytvoření Windows služby pro obsluhu laserového skeneru, vyhodnocovacího algoritmu a komunikace s OPC serverem. Dále se zabývá vytvořením aplikace pro zobrazení 3D modelu z naskenovaných dat a testování vytvořených aplikací. Práce je rozdělená do jednotlivých podkapitol, v kterých je podrobně popsán postup řešení jednotlivých bodů zadání. V první polovině je tato práce zaměřená zejména na analýzu a návrh řešení zadané problematiky. V druhé polovině je práce zaměřená na praktickou realizaci aplikací dle vytvořeného návrhu.

Klíčová slova: Microsoft .NET, C#, Microsoft Visual Studio, UML, OPC, Windows služba, WPF, MVVM, Návrhové vzory, Tcp/Ip protokol, Sick Lms 400, 3D model, Unit testy

Abstract

This diploma thesis deals with solving problems of creation of Windows service for laser scanner operation, evaluation algorithm and communication with OPC server. It also deals with creating an application for displaying a 3D model from scanned data and testing created applications. The thesis is divided into individual subchapters describing in detail the procedure of solving individual points of entry. In the first half, this work is mainly focused on the analysis and design of the solution of the given problems. In the second half, work is focused on the practical implementation of applications according to the created design.

Key Words: Microsoft .NET, C#, Microsoft Visual Studio, UML, OPC, Windows service, WPF, MVVM, Software design pattern, Tcp/Ip protocol, Sick Lms 400, 3D model, Unit tests

Obsah

Seznam použitých zkratek a symbolů

ASCII	–	American Standard Code for Information Interchange
COM	–	Component Object Model
DCOM	–	Distributed Component Object Model
LIDAR	–	Light Detection And Ranging
MVVM	–	Návrhový vzor Model-View-ViewModel
OLE	–	Object Linking & Embedding
OPC	–	Open Platform Communications
OPC DA	–	OPC Data Access
OPC UA	–	OPC Unified Architecture
PLC	–	Programovatelný logický automat
TCP/IP	–	Transmission Control Protocol/Internet Protocol
UML	–	Unified Modeling Language
WPF	–	Windows Presentation Foundation
XAML	–	Extensible Application Markup Language
XML	–	eXtensible Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato diplomová práce byla vytvořena ve firmě R.T.S. cs, spol. s.r.o. ve které již druhým rokem pracuji. Cílem této diplomové práce bylo vypracovat aplikaci pro detekci přepravních nádob pomocí laserového skeneru Sick Lms 400. Tato aplikace je použita v projektu, určeném na přenos přepravních nádob s díly pro automobilové převodovky, kde tato firma působila jako subdodavatel. K přenosu přepravních nádob je použit manipulátor, který je směřován na přesné souřadnice. Tyto souřadnice jsou získávány pomocí aplikace, vytvořené v rámci této práce.

V dnešní době, kdy je velmi populární průmyslová automatizace, pro zefektivnění a ulehčení práce zejména ve výrobě, dochází často k nahrazení lidské pracovní síly nějakým robotem nebo manipulátorem. Toto je právě příklad popisovaného projektu, kdy se jedná přímo o nahrazení operátorů, kteří překládali přepravní nádoby s díly do převodovek na dopravníkový pás vedoucí tyto přepravní nádoby do myčky. Jedná se tedy o projekt, zjednodušující a zefektivňující příslušnou část výroby.

Vytvořený projekt se skládá z řídícího automatizačního panelu (PLC), na kterém je spuštěna služba pro obsluhu laserového skeneru. Laserový skener je umístěn nad manipulátorem sloužícím pro přesun přepravních nádob, a tento manipulátor se pohybuje v rámci ochranné a nosné konstrukce do které se vkládají naložené palety s přepravními nádobami. Tento automatizační panel řídí komunikaci mezi službou obsluhující skener a manipulátorem pomocí OPC serveru. Celá procedura skenování a automatický přenos přepravních nádob z palety na přepravní pás myčky je řízen operátorem pomocí řídící aplikace spuštěné na automatizačním panelu.

Tato práce se zabývá postupem návrhu a následným vytvořením aplikace pro detekci výše popisovaných přepravních nádob. Hlavním úkolem je vytvoření Windows služby pro obsluhu laserového skeneru a WPF aplikace pro zobrazení naskenovaného prostoru pomocí 3D modelu. Součástí práce je vytvoření knihovny pro komunikaci se skenerem, vytvoření algoritmu pro zpracování a nalezení přepravních nádob z naskenovaných dat a vytvoření komunikace s OPC serverem.

Všechny částí této práce jsou rozdělené do jednotlivých kapitol. Druhá kapitola bude zaměřená na analýzu a návrh požadovaného řešení. Bude zde nastíněn návrh algoritmu zobrazený pomocí pseudokódu, a jednotlivé diagramy zobrazující funkci nejdůležitějších částí systému. V další kapitole bude podrobně popsán použitý laserový skener, jeho funkce a způsob vytvoření knihovny pro komunikaci sním. Ve čtvrté kapitole bude popsán postup při vytváření Windows služby. V následujících kapitolách bude práce zaměřená na popis realizace vyhledávacího algoritmu pro nalezení přepravních nádob a výpočet jejich souřadnic. V sedmé kapitole bude popsána komunikace mezi PLC panelem a Windows službou pomocí OPC protokolu. V následujících dvou kapitolách bude práce zaměřená na tvorbu aplikace pro zobrazení 3D modelu z naskenovaných dat a následně bude popsáno testování pomocí této aplikace a vytvoření Unit testů. V závěrečné kapitole bude popsáno dokončení aplikace, kde a jak byla nasazená a vytvoření konečné dokumentace pro předání projektu do výroby.

2 Analýza a návrh řešení

Na počátku tvorby každého projektu je velice důležité se zaměřit na co nejpodrobnější zanalyzování dané situace, a na základě teto analýzy vypracovat co nejpřesnější návrh aplikací. Během analýzy je potřeba se zaměřit zejména na zpracování všech funkčních a nefunkčních požadavků na návrh požadovaného softwaru. Z těchto požadavků se následně vytvoří návrh pomocí UML diagramů. Přesně tímto způsobem bylo postupováno při vytváření této práce, jak bude v následující kapitole podrobně popsáno.

2.1 Analýza požadavků

Cílem diplomové práce bylo vytvořit software, jenž má být součástí projektu pro automatizování procesu překládání přepravních nádob s díly na dopravníkový pás vedoucí do myčky dílů. Mezi hlavní úkoly aplikace patří detekování přepravních nádob a zjištění jejich souřadnic a rozměrů pro přesné navádění manipulátoru sloužícího k překládání přepravek. Následuje stručný popis problému z uživatelského hlediska:

Vytvořte aplikaci pro detekci přepravních nádob. Aplikace se bude skládat z Windows služby určené k detekování přepravních nádob na součástky pomocí laserového skeneru Sick Lms 400 a WPF aplikace pro zobrazení naskenovaných dat v 3D modelu. Po detekování přepravních nádob dojde k výpočtu souřadnic pro jednotlivé nádoby a poslání jich zpět pomocí zápisu do OPC. Výstupní souřadnice budou použité pro navedení 3D manipulátoru určeného k de-paletizaci a následného přeložení přepravních nádob na dopravníkový pás.

Na základě analýzy tohoto popisu problému byly vypracovány následující požadavky.

2.1.1 Funkční požadavky

V následující podkapitole budou popsané všechny funkční požadavky pro aplikaci, které byly vytvořené na základě komunikace s vedením firmy objednávající si tento software jako subdodávku pro celý projekt určený na překládaní přepravních nádob. Dále se požadavky upravovaly po konzultaci se zaměstnanci, kteří budou používat tuto aplikaci a s vedením zadavatelské firmy celého projektu. Výsledné funkční požadavky jsou následující:

- Běh aplikace na pozadí systému - Aplikace by měla běžet na pozadí a pracovat samostatně. Měla by být spustitelná skrze manažera služeb ve Windows.
- Komunikace s OPC serverem - Pro komunikaci mezi touto aplikací, PLC počítačem a manipulátorem bude použitý OPC server. Aplikace tedy musí být schopná se na něj korektně připojit, načíst si do něj všechny potřebné OPC položky určené pro komunikaci mezi všemi součástmi systému.
- Komunikace s laserovým skenerem Sick Lms 400 - Součásti aplikace musí být vytvořena knihovna, která bude zajišťovat komunikaci mezi aplikací a laserovým skenerem.

- Načtení konfiguračních souborů - Služba musí být schopna během startu si načíst všechny potřebné informace pro připojení jak k OPC, tak ke skeneru a také všechny potřebné parametry používané během provádění nalezení a výpočtů přepravních nádob.
- Zadání nového druhu přepravní nádoby - Služba musí umět zareagovat na přidání nového druhu přepravních nádob a to tak že si načte z konfiguračního souboru během startu parametry jednoznačně určující zadaný druh hledané přepravky.
- Možnost nalezení různých přepravních nádob - Aplikace bude umět nalézt různé druhy přepravních nádob.
- Zpracování příchozích naměřených dat - Příchozí data ze skeneru po provedení jednotlivých skenů chodí jako balík dat, obsahující parametry měření a naměřené vzdálenosti. Tyto vzdálenosti je nutné přepočíst na 2D souřadnice.
- Vytvoření 3D mapy povrchu vytvořené z jednotlivých skenů - V rámci složení 3D povrchu z jednotlivých skenů je třeba přepočítat 2D souřadnice na 3D souřadnice.
- Nalezení přepravních nádob - Po vytvoření naskenovaného 3D povrchu je nutné provést spuštění algoritmu pro nalezení jednotlivých přepravních nádob, tak aby bylo možné jednoznačně identifikovat jejich rozměry a jejich úlohu.
- Vypočítání parametrů pro jednotlivé přepravní nádoby - Dojde-li k nalezení přepravních nádob, tak bude následovat spuštění algoritmu pro výpočet jejich rozměrů, úhlu natočení a souřadnice průsečíku jejich uhlopříček.
- Odeslání výsledků - Zjištěné výsledky je nutné následně odeslat na OPC server a to tak, že se tyto parametry zapíšou do příslušných OPC položek.
- Zobrazování 3D mapy a naměřených výsledků - V rámci projektu bude vytvořená aplikace, umožňující si zpětně prohlédnout jakýkoliv skenovací proces a jeho výsledky.
- Opakovatelnost měření - Pro stejnou situaci musí být z různých skenovacích procesů stejné výsledky s tolerancí +/- 1,5 mm.
- Uložení dat z jednotlivých skenů - Pro kontrolu správnosti výsledku se bude ukládat naměřená data ve formátu CSV souboru.

2.1.2 Nefunkční požadavky

- Vytvoření Windows služby - Aplikace bude vytvořená formou Windows služby, tak aby byla schopna v reálném čase komunikovat mezi OPC serverem a laserovým skenerem. Musí být také schopna řešit všechny možné chybové stavy a být schopná kdykoliv vracet reálné výsledky.

- Dostupnost Windows služby - Služba musí být funkční a dostupná za každé situace, nemůže se stát, že dojde k výpadku komunikace s ní.
- Umístění - Služba bude umístěna a spuštěna na automatizačním panelu sloužícímu k obsluze manipulátoru pro de-paletizaci.
- Zálohování - Aplikace bude na předem určeném místě na disku udržovat naměřená data z posledních sto skenů.

2.1.3 Podrobný popis funkcionality jednotlivých částí systému

- Služba pro obsluhu skeneru Sick Lms 400

  Služba se spustí automaticky po zapnutí počítače. Následně si načte všechny potřebné konfigurační soubory a připojí se ke skeneru pomocí TCP/IP protokolu. Současně se připojí k OPC serveru a načte si všechny položky a jejich hodnoty.

  Když proběhnou všechna nastavení správně, nastaví do OPC na položku „aplicationReady“ hodnotu na „true“. Od této doby bude v pohotovostním režimu a bude čekat na nějakou změnu v OPC.

  Služba bude průběžně kontrolovat, zdali nedošlo k nějaké chybě nebo odpojení skeneru. Když přijde z OPC požadavek, že se má začít skenovat, tak pošle do skeneru zprávu na odstartování „triggerového“ skenování (skenování po jednotlivých skenech řízené hardwarovým spouštěčem) a následně zapíše do OPC, že aplikace začíná skenovat.

  Když se začne pohybovat manipulátor se skenerem, tak každého půl milimetru dojde k oskenování plochy pod skenerem. Tyto data se přepočtou na 3D souřadnice. Naskenované data přestanou přicházet, když se ukončí pohyb manipulátoru se skenerem. Následně se čeká až se v OPC shodí signál, že má probíhat měření a aplikace pošle ukončovací zprávu do skeneru.

  Následně se provede vyhodnocení naskenovaných dat, z nich se najdou přepravní nádoby nejvýše položené v každém sloupci a spočítají se souřadnice jejich průsečíků a jejich rozměry. Dále se provede vyhodnocení, zdali jsou výsledky správné a zapíšou se do OPC současně s položkou „resultsValid“, která značí že jsou platné výsledky. V opačném případě se nezapíšou výsledky do OPC, ale zapíše se tam položka „error“, která znamená že došlo k nějaké chybě během měření. Po ukončení vyhodnocení je služba připravena na další měření.

- Aplikace pro zobrazení a kontrolu provedených skenování

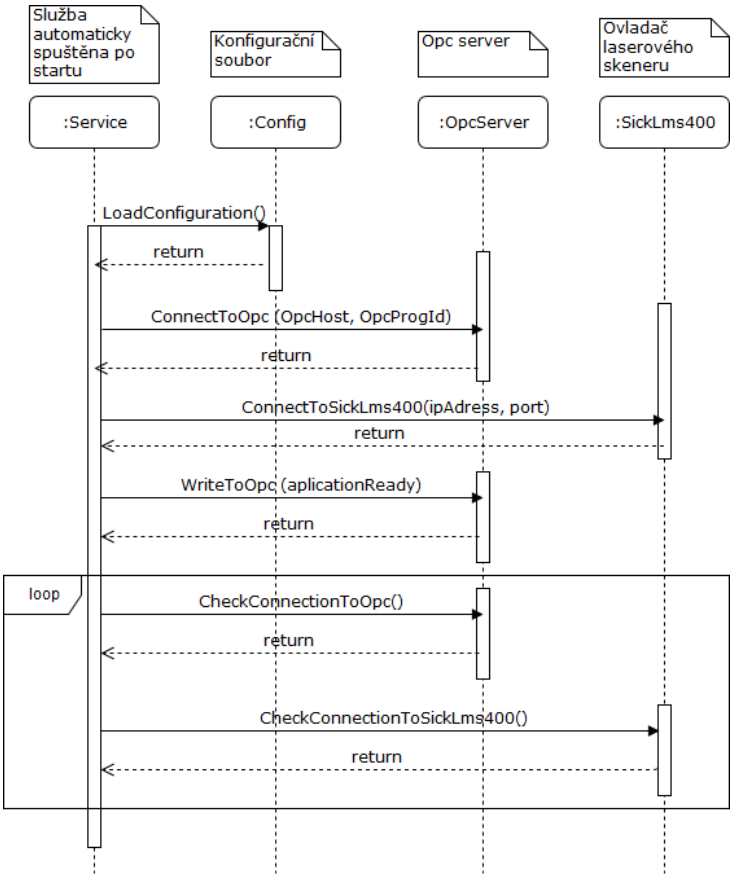
  Aplikace bude sloužit pro zobrazení naskenovaných dat pomocí služby popsané výše. Data se budou zobrazovat jako 3D model, kterým lze různě pohybovat. Načtení dat se bude provádět pomocí otevření libovolného souboru s naskenovanými daty. Po stisku tlačítka určeného na filtraci zobrazených dat dojde k průchodu dat a nalezení přepravních nádob

pomocí stejných algoritmů použitých při vyhodnocování ve službě. Výsledky pro jednotlivé nalezené nádoby se zobrazí v aplikaci, tak aby šlo zkontrolovat jednotlivé skeny, zdali se vyhodnocují správně a nedochází v jejich detekci a výpočtu k určitým chybám.

2.2 UML diagramy

V této kapitole bude zobrazená plánovaná funkčnost aplikace pomocí jednotlivých UML diagramů [1]. Každý diagram bude následně podrobně popsán.

2.2.1 Sekvenční diagram znázorňující spuštění služby

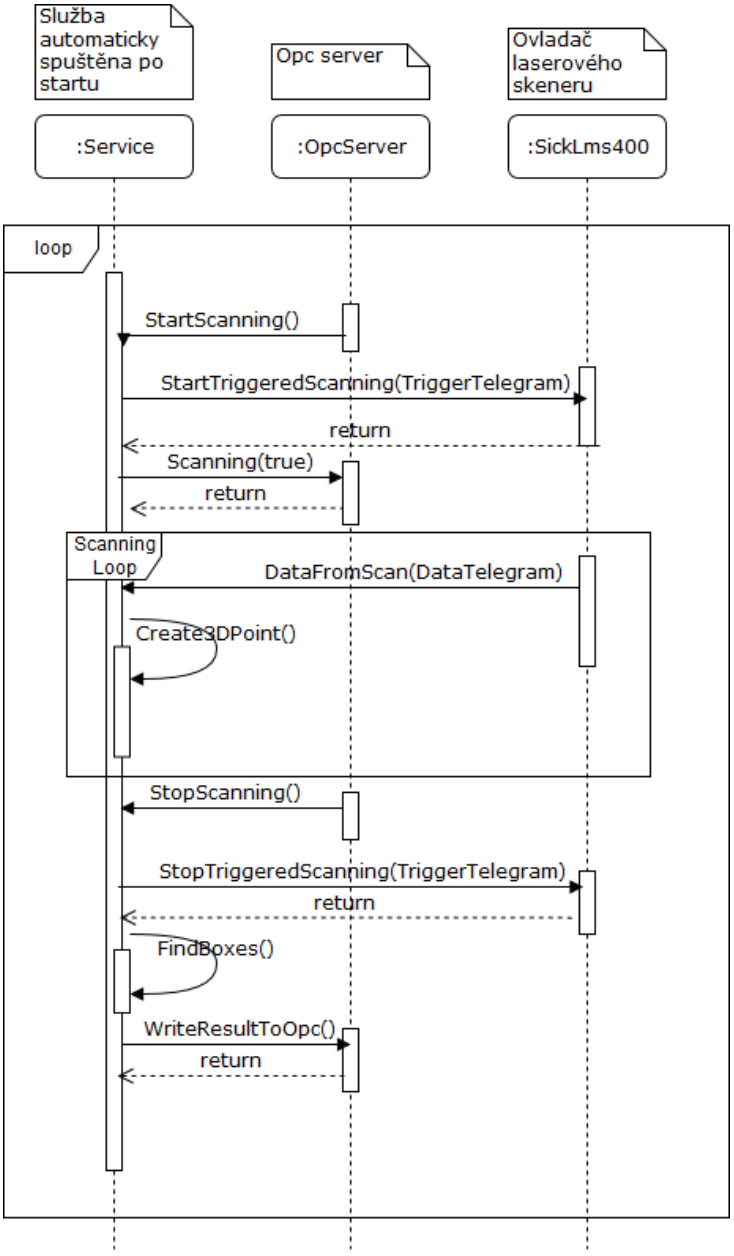


Obrázek 1: Start Windows služby

Jak je zobrazeno na tomto sekvenčním diagramu 1, tak služba pro obsluhu skeneru Sick Lms 400, se automaticky spustí po startu počítače. Po spuštění si nejdříve načte z konfiguračního souboru všechny potřebné údaje k připojení na OPC server a všechny potřebné údaje a nastavení k připojení k laserovému skeneru. Následně dojde k připojení na OPC server a načtení všech OPC položek na něj. Poté se služba pokusí připojit na laserový skener pomocí TCP/IP protokolu. Když je připojení úspěšné tak se pošlou na skener zprávy vyžadující změnu nastavení skenování, podle parametrů načtených z konfiguračního souboru. Když jsou úspěšně dokončeny všechny

tyto nastavení a připojení tak dojde k zapsání do OPC na položku „aplicationReady“ na „true“, v opačném případě se na tuto položku zapíše „false“. Následně se spustí vlákno služby, které bude v celém průběhu běhu služby kontrolovat, zdali nedošlo k chybě v připojení na OPC server nebo na laserový skener.

2.2.2 Sekvenční diagram znázorňující skenování

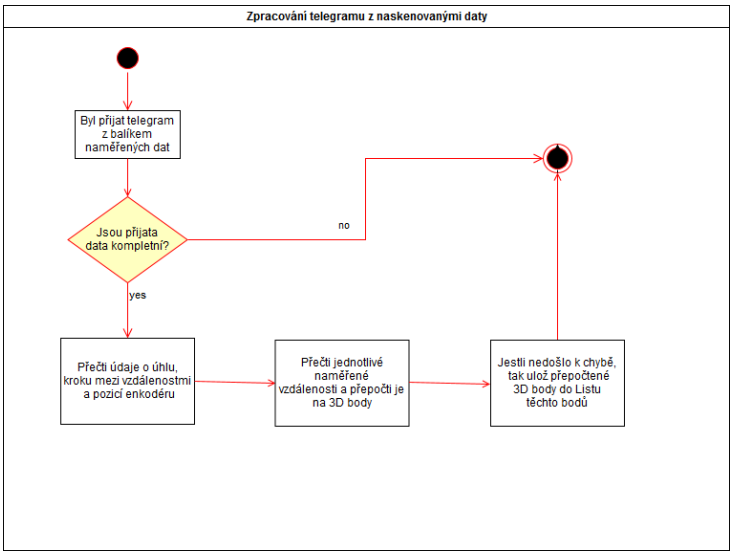


Obrázek 2: Skenovací proces

V tomto diagramu 2 je znázorněno, jak se služba bude chovat po načtení konfigurace a provedení připojení ke skeneru a OPC serveru popsaném v předchozím diagramu. Aplikace čeká na

změnu některé z položek na OPC serveru. V případě že se zapíše v OPC na položce označující start skenování zapíše „true“, tak dojde k poslání zprávy na laserový skener, že se odstartuje skenování řízené hardwarovým spouštěčem (trigger). Následně z každým pohybem manipulátoru, na kterém je umístěn laserový skener dojde k zaslání balíku dat z naskenovanými daty. Tyto balíky dat budou chodit až do momentu, než dojde k ukončení pohybu manipulátoru. Každý datový balíček příchozí ze skeneru je službou průběžně zpracováván a převáděný na List 3D bodů. Měření je ukončeno v momentu, kdy se zapíše na OPC položku označující start skenování „false“. Následně služba pošle na skener zprávu o ukončení skenování a začne zpracovávat všechny naskenovaná data. Z těchto dat nalezne přepravní nádoby a propočítá jejich rozměry, souřadnice průsečíku a úhel natočení přepravních nádob. Tyto hodnoty následně zapíše do OPC na příslušné položky.

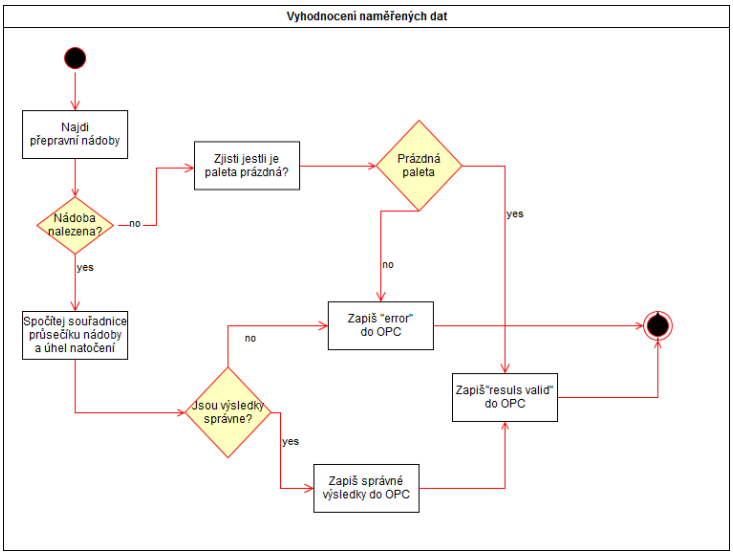
2.2.3 Diagram znázorňující zpracování přijatého balíku s naskenovanými daty



Obrázek 3: Zpracování balíku dat

V tomto diagramu 3 je znázorněné, jak bude probíhat zpracování přijatých balíku dat z jednotlivých skenů zaslané laserovým skenerem. Zpracování začne tím, že služba zašle balík dat obsahujících jak parametry skenování, tak balík naskenovaných vzdáleností. Následně dojde ke kontrole, zdali přijatý balík dat je kompletní a bez chyb. Když dojde k nějaké chybě tak se ukončí zpracování a vyhodí se příslušná chyba, když je vše v pořádku tak se pokračuje ve zpracování. Na základě úhlu a kroku mezi jednotlivými vzdálenostmi dojde k propočtu na souřadnice v ose Y a Z. Souřadnice v ose X se získá z položky polohy manipulátoru, která přijde s každým skenem. Tento přepočet pokračuje, dokud nejsou zpracovány všechny hodnoty vzdálenosti, které byly přijaty. Nakonec se zkontroluje, jestli nedošlo k nějaké chybě, všechny 3D souřadnice se uloží do Listu těchto hodnot a ukončí se daná aktivita.

2.2.4 Diagram vyhodnocení naměřených dat



Obrázek 4: Vyhodnocení výsledků

Po zpracování všech přijatých naměřených hodnot popsané v předchozím diagramu, dojde ke zpracování všech 3D bodů. Jak je zobrazeno na tomto diagramu 4 , tak nejdříve dojde k použití algoritmu pro detekci přepravních nádob. Nedojde-li k nalezení žádné přepravní nádoby, tak se následně provede kontrola, zdali byla skenována již prázdná paleta. V případě, že byla již paleta prázdná tak dojde k zápisu do položky v OPC „reslutsValid“ na „true“. V opačném případě došlo během vyhodnocování nejspíše k nějaké chybě, tak se zapíše do OPC na tuto položku „error“.

V případě že dojde k nalezení přepravních nádob, tak následně dojde k propočítání jejich rozměrů, souřadnic průsečíků a úhlu natočení přepravní nádoby. Následně se z vypočtených rozměrů provede kontrola, jestli jsou správné výsledky. Opět se provede v případě, když proběhne všechno v pořádku zápis výsledků do OPC a zápisu „true“ na položku „resultsValid“ v OPC. Když dojde k nějaké chybě tak se zapíše opět „error“.

2.3 Použité technologie

Jelikož má být aplikace pro obsluhu laserového skeneru v nepřetržitém provozu, a její funkcionalita má být samostatná, tak se nabízí použití Windows služby. Windows služba bude tedy naprogramována v programovacím jazyce C# na platformě Microsoft .NET ve verzi 4.6.1 . K vytvoření bude použito Microsoft Visual Studio 2013. Bude zde využitá technologie vláken, a to tak že jedno vlákno bude použito jako pracovní vlákno, zachycující všechny změny v položkách na OPC serveru. Druhé vlákno bude sloužit pro nepřetržitou kontrolu, zda nedošlo k výpadku spojení jak se skenerem, tak s OPC serverem.

Pro testování bude použit testovací Framework NUnit. Slouží k vytváření Unit testů, označování a vylučování tříd a metod k testování. Mezi jeho výhody patří, že lze označit metody, které se spustí před testem, po testu nebo po celé či před celou třídou.

Ke komunikaci s PLC bude sloužit již dříve zmiňovaná technologie OPC, která je založená na OPC protokolu a poskytuje jednotné rozhraní pro výměnu dat mezi aplikacemi a koncovými zařízeními prostřednictvím OPC Serveru.

Pro zobrazení naskenovaných dat a výsledků vyhodnocovacího algoritmu bude vytvořena WPF aplikace [2]. Je to knihovna tříd pro tvorbu grafického rozhraní, jenž je součástí Microsoft .NET Frameworku. Jedná se o moderní technologii, nahrazující Windows Forms. Lze v ní snadno oddělit funkčnost od vzhledu aplikace, která se píše pomocí značkovacího jazyka XAML. Tato aplikace bude postavena na návrhovém vzoru MVVM, který je popsán podrobněji níže.

2.4 Návrhové vzory

Pro WPF aplikaci byl vybrán návrhový vzor MVVM (Model-View-ViewModel) [3]. Tento návrhový vzor je založený na velmi dobrém oddělení logiky aplikace od uživatelského rozhraní. Technologie WPF byla vytvořena a později přizpůsobena, aby se v ní tento návrhový vzor používal velmi jednoduše. V tomto návrhovém vzoru se se používá „data binding“ a „commanding“, což je náhrada za událostmi řízeném uživatelském rozhraní. Díky oddělení logické vrstvy od uživatelského rozhraní je také aplikace postavena na tomto návrhovém vzoru velmi dobře testovatelná.

Hlavním principem MVVM je vytvořit třídu, která udržuje stav aplikace. Jedná se o tzv. ViewModel, který si dotazuje uživatelské rozhraní a podle něho vykresluje ovládací prvky. V opačném případě, když zadá uživatel do uživatelského rozhraní nějaké nové údaje, tak se automaticky převedou do ViewModelu, a to je umožněno zejména díky již zmiňovanému „data bindingu“, díky kterého lze deklarativně napojit uživatelské rozhraní na ViewModel.

Třída ViewModelu je základním prvkem tohoto návrhového vzoru, protože poskytuje všechny data pro uživatelské rozhraní. Uživatelské rozhraní se nazývá View a pomocí něho jsou veškerá data zobrazeny uživateli. View je založený na dvou hlavních prvcích.

Prvním je „ObservableCollection<T>“, který sleduje a upozorňuje, když je přidán nebo odebrán nějaký prvek. Druhým prvkem je „INotifyPropertyChanged“, který popisuje nějakou událost, která nastala, když se změnila některá z vlastností ViewModelu.

Návrhový vzor, který je zobrazen na následujícím obrázku 5 se skládá z těchto tří vrstev:

- Model

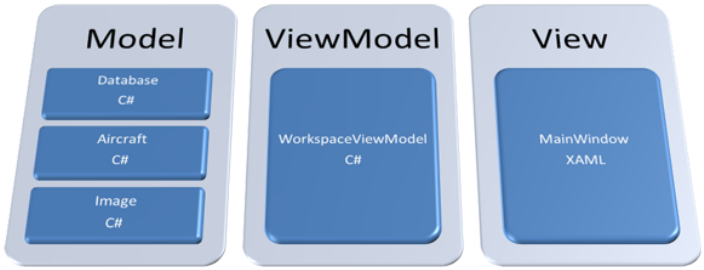
  Slouží k popisu dat, se kterými pracuje aplikace. V případě, že zdrojem dat je například databáze, tak třídy, které mapují tabulky této databáze jsou Modely. Aby byl zachován princip tohoto návrhového vzoru, tak model nesmí o stavu ovládacích prvků nic vědět.

- View

Je to prvek, který reprezentuje uživatelské rozhraní. Je popsán v jazyce XAML. Jedná se o jakékoliv okno aplikace, ovládací prvek nebo stránku.

- ViewModel

  Drží stav a data aplikace a zajišťuje propojení mezi předchozími dvěma vrstvami. Ovládací prvky čerpají z něho svůj obsah a jsou sním propojeny pomocí „data bindingu“. Lze v něm provádět filtrování dat v závislosti na stavu aplikace.



Obrázek 5: Návrhový vzor MVVM [4]

2.5 Dostupnost aplikací

Jelikož jsou tyto aplikace určené pro použití ve výrobě na automatizačních panelech tak jsou uzpůsobené na provoz pro konkrétní zadanou specifikaci:

Automation Panel 15.6"HD TFT

- Panel PC 2100
  - Intel Atom E3845 1,91 GHz
  - Quad core
  - 4 GB SDRAM
  - For Automation Panel 923/933
- Windows Embedded Standard 7 SP1
  - 64 bit
  - Service Pack 1
  - English
  - PPC2100

Na základě této specifikace bylo rozhodováno, jakým způsobem budou aplikace fungovat, jak bude probíhat výpočet výsledných souřadnic přepravních nádob, a jak se budou zobrazovat naskenovaná data, vzhledem k poměrně slabšímu výkonu tohoto panelu oproti běžným počítačům používaných k vývoji softwaru. Obě předem popsané aplikace budou nasazené na tomto zařízení.

2.6 Návrh vyhodnocovacího algoritmu

Algoritmus pro vyhodnocení naskenovaných dat bude probíhat v postupných krocích v několika následujících funkcích:

- Nejdříve se množství naskenovaných dat zmenší o body, které pocházejí z prostoru mimo požadovaný skenovací prostor. Jelikož se jedná o laserový skener, tak by mohlo těmito body dojít ke zkreslení výsledků, protože může nastat nežádoucí odraz laserového paprsku od nějaké překážky při návratu do skeneru. Filtrace se docílí tak, že se v konfiguračním souboru nastaví rozměry detekční zóny a následujícím algoritmem se provede kontrola, zda bod leží v zóně. Průchod se provede tak, že se projdou jednotlivé skeny a zkontrolují se všechny jeho body. Jestli se bod nachází v detekční zóně tak se uloží do listu těchto 3D bodů opět pro jednotlivé skeny, viz. výpis kódu: 1.

- Dalším krokem bude projít všechny uložené body z předchozího kroku pro jednotlivé skeny, a pokusit se najít v nich případné vrcholy přepravních nádob, viz. výpis kódu: 2.

- Následně se z nalezených levých a pravých potencionálních vrcholů přepravních nádob vypočtou parametry jako je šířka, výška a uloží se tyto parametry jako objekt reprezentující fragment potencionálně nalezené přepravní nádoby, viz výpis kódu: 3.

- Tyto nalezené objekty následně projdou kontrolou a dojde k vyřazení objektů, které nejsou součástí přepravních nádob, viz výpis kódu: 4.

- Po provedení této kontroly nám zůstanou pouze objekty z jednotlivých skenů, které tvoří části přepravních nádob, uložených na hoře ve sloupcích s přepravními nádobami.

  V dalším kroku se projdou všechny objekty popsané výše, a dojde k poskládání jednotlivých přepravních nádob z těchto objektů, viz výpis kódu: 5.

- V posledním kroku se provede ještě kontrola nalezených přepravních nádob, jestli odpovídají jejích rozměry hledaným typům přepravních nádob a následně se vypočítají jejich parametry, viz výpis kódu: 6.

Algorithm 1 Algoritmus zobrazený v pseudokódu pro přetřídění bodů na ty co se nachází v detekční zóně.

Input: List<3DPoint> *points*, object *detectionZone*
Output: List<3DPoint> *pointsFromZone*

```
1: for p in points do
2:    Vezmi p a zjisti jestli se nachází v : detectionZone
3:    if p se nachází v : detectionZone then
4:       Přidej p do: pointsFromZone
5:    else
6:       Zahoď
7:    end if
8: end for
```

Algorithm 2 Algoritmus zobrazený v pseudokódu pro nalezení vrcholů přepravních nádob.

Input: List<3DPoint> *pointsFromZone*, int *ZTolerance*
Output: float *minY*, float *maxY*, float *minZ*

```
1: for p in pointsFromZone do
2:    Vezmi p a zjisti jestli se nejedná o nějaký nežádoucí odražený bod
3:    if p je nežádoucí odražený bod then
4:       Zahoď
5:    else
6:       Pokračuj
7:    end if
8:    Najdi nejméně vzdálený bod od skeneru minZ
9:    if p.Z < minZ then
10:      minZ = p.Z
11:   else
12:      Pokračuj
13:   end if
14: end for
15: for p in pointsFromZone do
16:   Vezmi p a najdi nejméně vzdálený bod od skeneru z levé a pravé strany skenu
17:   if p.Z - minZ <= ZTolerance then
18:      Pokračuj
19:      if p.Y < minY then
20:         minY = p.Y
21:      end if
22:      if p.Y > maxY then
23:         maxY = p.Y
24:      end if
25:   else
26:      Pokračuj
27:   end if
28: end for
```

Algorithm 3 Vytvoření objektů reprezentující fragment přepravní nádoby v pseudokódu.

Input: float $minY$, float $maxY$, float $minZ$, float $xPosition$
Output: List<object> $containerFragments$

1: $width = maxY$ - $minY$
2: $height = minZ$
3: $leftPoint = (\ minY, minZ$
4: $rightPoint = (\ maxY, minZ$
5: Vytvoř nový objekt $containerFragment$ z: ($leftPoint$, $rightPoint$, $width$, $height$, $xPosition$)
6: Přidej $containerFragment$ do: $containerFragments$

Algorithm 4 Kontrola objektů reprezentujících fragment přepravní nádoby v pseudokódu.

Input: List<object> $containerFragments$, int $ZTolerance$
Output: List<object> $checkedObjects$

1: Projdi $containerFragments$ a najdi objekt který je nejméně vzdálený od skeneru.
2: **for** o in $containerFragments$ **do**
3: **if** $o.Height < minObject.Height$ **then**
4: $minObject = o$
5: **else**
6: Pokračuj
7: **end if**
8: **end for**
9: Projdi $containerFragments$ a zkontroluj jejich vzdálenost od skeneru.
10: **for** o in $containerFragments$ **do**
11: **if** $o.Height$ - $minObject.Height <= ZTolerance$ **then**
12: Přidej o do: $checkedObjects$
13: **else**
14: Zahoď objekt
15: **end if**
16: **end for**

Algorithm 5 Vytvoření objektu reprezentující přepravní nádobu v pseudokódu.

Input: List<object> *checkedObjects*, int *xTolerance*, int *minCount*
Output: List<object> *containerObjects*

```
Projdi checkedObjects a vytvoř z nich jednotlivé přepravní nádoby.
Vytvoř containerObject z: ( checkedObjects.First)
for o in checkedObjects do
    if o.xPosition - containerObject.lastXPosition < xTolerance then
        Přidej o do: containerObject
    else
        Zkontroluj počet objektů podle minCount
        if containerObject.Count > minCount then
            Přidej containerObject do: containerObjects
            Vytvoř containerObject z: ( o)
        else
            Zahoď containerObject
        end if
    end if
end for
```

Algorithm 6 Kontrola objektů reprezentujících přepravní nádoby a výpočet jejich parametrů v pseudokódu.

Input: List<object> *containerObjects*, int *widthTolerance*, int *lengthTolerance*
Output: List<object> *calculatedContainerObjects*

```
Projdi containerObjects a vypočti jejich rozměry.
for object in containerObjects do
    minY ⇐ minval containerObjects.leftPoint.Y
    maxY ⇐ maxval containerObjects.rightPoint.Y
    minX ⇐ minval containerObjects.xPosition
    maxX ⇐ maxval containerObjects.xPosition
    Width = minY - maxY
    Length = minX - maxX
    if Width <= widthTolerance and Length <= lengthTolerance then
        Přidej o do: calculatedContainerObjects
    else
        Zahoď objekt o
    end if
end for
Projdi calculatedContainerObjects a vypočti jejich parametry.
for o in calculatedContainerObjects do
    Najdi průsečíky objektu o
    Vypočti souřadnice průsečíků
    Spočítej úhel natočení přepravní nádoby o
    Zapiš výsledky do OPC
end for
```

3 Vytvoření ovladače pro komunikaci s laserovým skenerem Sick Lms 400

Po vytvoření návrhu bylo následně přistoupeno k realizaci zadaného úkolu. K tomu bylo nutné nejdříve se seznámit s laserovým skenerem, zjistit, jak funguje a vytvořit knihovnu pro komunikaci mezi ním a Windows službou.

V projektu byl použit na žádost zadavatele laserový skener od německé firmy Sick. Tato firma se zabývá již několik let vývojem laserových zařízení k detekci předmětů nebo různým měřením. Firma je jedním z největších výrobců inteligentních senzorů a zaměřuje se především na optické senzory určené k průmyslové automatizaci a tvorbu bezpečnostních systémů.

3.1 Popis skeneru

Použitý skener pochází z řady Lms4xx, která je zaměřená zejména na velkou přesnost měření, jenž je dosažená díky úhlovému kroku mezi jednotlivými body, který je nastavitelný od 0.1333 do 1°. Přesnost měření u těchto skenerů je podobná jako u radaru, jen se využívá jiný princip skenování. Tento skener patří mezi 2D LIDAR senzory [5], [6], což znamená že využívá metodu měření vzdálenosti na základě výpočtu doby šíření impulsu laserového paprsku, který se odrazí od snímaného objektu. LIDAR metoda se používá zejména pro měření vzdálenosti, mapování terénu, měření atmosférických jevů a jiné. Výsledkem skenování je mračno bodů, které se po upravení dá interpolovat do podoby 3D modelů například budov a jiných objektů. Na tuto metodu se dají také aplikovat různé filtry, díky níž je možné z mračna naskenovaných bodů získat digitální model terénu.

Skener je tedy přesného typu Lms 400-2000 [7], který je přímo určen na de-paletizační úkony, což je v tomto projektu velmi důležité. Má dynamický rozsah měření od 0,7 do 3 metrů, který je v našem případě dostačující. Skener má vysoké úhlové rozlišení a vysokou frekvenci snímání, díky které je skenování velice rychlé a spolehlivé, viz. tabulka s vlastnostmi skeneru: 1

Tento skener dále poskytuje různé digitální filtry, které se dají libovolně kombinovat a slouží pro optimalizaci naskenovaných dat. Tyto digitální filtry jsou následující:

- Mean filter - tento filtr vytvoří z určitého počtu skenů aritmetický průměr. Počet skenů se nastavuje podle požadavků a až po přijatí počtu těchto skenů, skener vrátí naměřené hodnoty. Tímto filtrem dojde k zamezení špatných odrazů nebo jiných šumů, avšak v našem případě kvůli požadované rychlosti skenování tento filtr nebyl použit.
- Range filter - v tomto případě se nastaví minimální a maximální dosah skeneru z kterého bude vracet data.
- Edge filter - tímto filtrem se dá nastavit krajní body, které již nebudou posílány zpět.
- Median filter - vytvoří medián z určitého počtu skenů, který se opět zadá. Tento filtr opět nebyl použit z důvodu rychlosti skenování.

Tabulka 1: Hlavní parametry laserového skeneru Sick Lms 400

Vlastnost	**Parametr**
Světelný zdroj	Viditelné červené světlo (650 nm)
Úhel zorného pole	70°
Frekvence snímání	300 Hz - 500 Hz
Úhlové rozlišení	0,1° - 1°, volitelné
Pracovní oblast	0,7 m - 3 m
Doba odezvy	$\geq$ 2 ms
Systematická chyba	$\pm$ 4 mm
Statistická chyba	3 mm
Ethernet rozhraní	TCP/IP
Sériové rozhraní	RS-232
Odrazivost objektu	4,5 % - 100 %

3.2 Princip skeneru

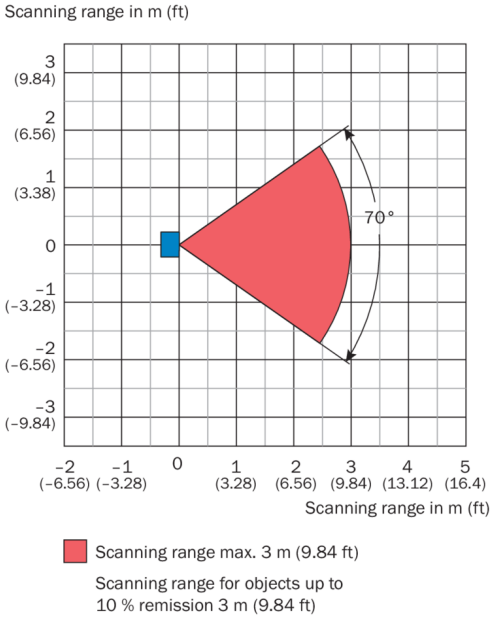
Funkce laserového skeneru, který funguje na principu dálkoměru je velmi podobný jako u radaru. Skener vyšle impuls z nějakého zdroje, v tomto případě laserové diody s příslušnou optickou soustavou a čeká, než se vyslaný impuls odrazem od nějaké překážky vrátí zpět do skeneru. Odražený signál je ve skeneru detekován pomocí lavinové foto-diody. Touto foto-diodou je světelný impulz převeden zpět na elektrický proud. Výsledná vzdálenost se následně vypočte z rozdílu času, kdy byl impuls vyslán a kdy se impuls vrátil.

Tento model laserového skeneru obsahuje zrcadlový hranol, který se neustále otáčí kolem své vlastní osy a jeden měřící cyklus je dokončen během jednoho úplného otočení hranolu kolem své vlastní osy. Díky tomuto hranolu je skener schopný zachytit skenovaný prostor před sebou v úhlu až 70°. Skenovací prostor je zobrazený na následujícím obrázku. 6

Na kvalitě naskenovaných dat velmi závisí odrazivost neboli reflektivita skenovaného objektu. Každý materiál má rozdílnou odrazivost a ta je velice důležitá pro sílu optického signálu vyslaného zpět do skeneru. Čím je odrazivost vyšší, tím je skenování přesnější. U tohoto skeneru by měla být odrazivost u skenovaných objektů alespoň 10 %. V našem případě skenované přepravní nádoby jsou z kovového materiálu, který má vesměs dobrou reflektivitu pohybující se mezi 110 až 150 %, viz tabulka [8], ale velký vliv nato má ještě barva. Černá barva má velmi malou reflektivitu, takže se může stát, že při vyšším znečištění přepravní nádoby může být reflektivita a tím i přesnost měření horší, avšak v našem případě reflektivita neklesala pod 50 %, což je dostatečná hodnota.

3.3 Princip komunikace se skenerem

Komunikace se skenerem je možná buď dodávaným softwarem SOPAS od firmy Sick, sériovým rozhraním RS-232 nebo Ethernetovým rozhraním. Z počátku byl použitý dodávaný software SOPAS na první zkoušky skenovaní a zjištění reflektivity přepravních nádob a uvážení zdali



Obrázek 6: Diagram pracovní oblasti laserového skeneru Sick Lms 400[9]

se tento skener hodí na námi řešený problém. Tato aplikace také stačí pro základní nastavení skenovacích parametrů skeneru.

Po počátečním seznámení a ověření vhodnosti vybraného zařízení bylo potřeba vytvořit knihovnu pro komunikaci sním. Knihovna byla vytvořena v jazyce C# a pro komunikaci bylo vybráno ethernetové rozhraní kvůli vyšší rychlosti, jelikož sériové rozhraní dostačuje pouze k nastavení zařízení, a ne ke skenování v reálném čase. Byla tedy vytvořena komunikace pomocí TCP/IP protokolu na principu klient-server. Skener byl v tomto případě jako server a vytvářená knihovna jako klient posílající na server zprávy.

Ke komunikaci na tomto zařízení slouží tzv. telegramy do kterých se zapouzdří požadované příkazy nebo odpovědi a tyto telegramy jsou dále zapouzdřené do komunikačních paketů. Tyto telegramy, komunikace se skenerem a všechny operační instrukce jsou popsané v této dokumentaci [10]. K vytvoření Ethernetové komunikace byla použitá třída TcpClient, která je součástí Microsoft .NET Frameworku. K vytvoření připojení je třeba již jen znát IP adresu zařízení a číslo jeho komunikačního portu. Skener má z výroby nastavenou IP adresu na: 192.168.0.1, která se dle požadavků dá zasláním příslušného příkazu na skener změnit. TCP/IP port je od výroby přiřazen na hodnotu 2111, jenž je primárně určen ke konfiguraci a 2112, který je určen pro reálné skenování, a proto bude použit ve vytvářené aplikaci. Tyto porty se nijak nedají měnit a ani to není potřeba. Pro připojení se tedy vytvoří nová instance třídy TcpClient a zavolá se na ni metoda „Connect(IP, Port)“, kde se zadá jako parametr IP adresa a port skeneru. Následně se vytvoří nové vlákno, určené pro poslouchání tohoto komunikačního kanálu pro příjem odpovědí od skeneru.

Pro zaslání příkazu nebo přijímání odpovědi ze zařízení je třeba vytvořit telegram, jehož

struktura je zobrazena na následujícím obrázku 7. Na obrázku jde vidět, že telegram je zapouzdřen do rámce. První část tohoto rámce se skládá z TCP/IP rámce, který je definovaný podle druhu přenosu. Pro vytvoření tohoto rámce byla použitá třída NetworkStream, která je součástí Microsoft .NET Frameworku. Při zápisu pomocí této třídy se použije metoda „Write(byte[] buffer, int offset, int size)“, kde se do prvního parametru zadá telegram převedený na pole bytů, offset, který je standardně nastaven na 0 a délka posílaného telegramu. Další část počátečního rámce je tvořená z počátečních startovacích symbolů, a délky samotného telegramu. Následuje samotný telegram, který obsahuje příkaz nebo zprávu zasílanou na skener a skládá se z ASCII symbolů. Tento telegram začíná prvními třemi symboly, které definují, o jaký typ zprávy se jedná:

- sMN - Začátek telegramu pro poslání zprávy na spuštění skenování.
- sMA - Začátek telegramu poslaný zpět ze skeneru, že zaslaná žádost byla úspěšně doručena na skener a je zpracovávána.
- sAN - Počátek telegramu z odpovědí ze skeneru, že byla zaslaná žádost zpracována a následuje potvrzení nebo kód chyby která nastala.
- sFA - Počátek telegramu zaslaný ze skeneru oznamující chybu, která nastala na skeneru.
- sWN - Těmito symboly začíná zpráva zaslána na skener ze žádosti o změnu nějaké proměnné.
- sWA - Těmito symboly začíná zpráva poslána ze skeneru z potvrzením, že žádost o změnu proměnné se zpracovává.
- sMN - Počátek telegramu posílaného na skener obsahující žádost na změnu nastavení skeneru.
- sMA - Začátek telegramu zaslaného ze skeneru z potvrzením, že žádost na změnu nastavení se zpracovává.

Za počátečními symboly popsanými výše následuje již konkrétní zpráva z požadavkem nebo odpovědi ze skeneru. Telegram je následně zakončený kontrolním součtem. Ten se vypočítá pomocí operátoru XOR pro každý bajt dat. Zdrojový kód pro vytvoření telegramu vypadá následovně:

```
public byte[] CreateInstruction (string instruction) {
      string Telegram = "";
      byte[] head = { 0x02, 0x02, 0x02, 0x02 }; //pocatecni bity zacinajici
          kazdou zpravu

      char[] instructionSymbols = instruction.ToCharArray();
```

Telegrams on the Ethernet interface

	Frame						Telegram	Frame	
Code	TCP/IP Start Frame	STX	STX	STX	STX	Telegram length	Data (see *11.2 on page 80*)	Check-sum	TCP/IP Stop Frame
Length (byte)	Defined by the trans-mission	1	1	1	1	4	≤2495	1	Defined by the trans-mission
Description		Start of text character				Data length without CS, Motorola format	Binary encoded. The length is dependent on the previous send telegram.	See *"Calcu-lation of the check-sum"* further below	

Tab. 15: Frame for the telegrams on the Ethernet interface

Obrázek 7: Struktura telegramu pro komunikaci se skenerem

```
        // Vypocita se kontrolni soucet
        int checkSum = instructionSymbols[0];
        for (int i~= 1; i~< instructionSymbols.Length; i++) {
           checkSum ^= instructionSymbols[i];
        }
        string Checksum = checkSum.ToString("X");

        byte[] telegramLength = BitConverter.GetBytes(instruction.Length);
        if (BitConverter.IsLittleEndian) {
           Array.Reverse(telegramLength);
        }
        Telegram = Encoding.ASCII.GetString(head, 0, head.Length);
        Telegram += Encoding.ASCII.GetString(telegramLength, 0, telegramLength.
            Length);
        Telegram += instruction;
        Telegram += Checksum;

        return Encoding.ASCII.GetBytes(Telegram);
    }
```

Výpis 1: Zdrojový kód pro vytvoření telegramu

Komunikace například pro spuštění skenování mezi počítačem a skenerem vypadá následovně:

- Vyšle se požadavek na skener: sMN mLRreqdata 0020
- Po provedení kódu na vytvoření instrukce zobrazeného výše se odešle do skeneru tato zpráva: (STX)(STX)(STX)(STX)(délka)sMN mLRreqdata(Kontrolní součet)

- Skener zašle potvrzení, že přijal zprávu a zpracovává jí: sMA mLRreqdata
- Skener pošle odpověď, že byl příkaz zpracován: sAN mLRreqdata 00000000
- Následně skener začne cyklicky posílat zprávy obsahující naskenované data

Pomocí tohoto principu byla dále knihovna rozšířena o všechny potřebné příkazy, které jsou podrobně popsané v dokumentaci. Jednalo se o příkazy pro nastavení komunikace se skenerem, nastavení parametrů pro měření a nastavení módu zajištujícího měření pomocí hardwarové spouště. Ve vlákně určeném pro přijímání zpráv zaslané ze skeneru, byly vytvořeny metody rozpoznávající jednotlivé odpovědi pro kontrolu, zdali došlo ke korektnímu zpracování zaslaných příkazů a dále také ke zpracování všech chyb, které můžou během komunikace nastat. V tomto vlákně byla vytvořená také funkce na zpracování samotného balíku z naskenovanými daty. Zpracování naskenovaných dat bylo vytvořeno podle principu znázorněného v přiložené dokumentaci.

4 Vytvoření Windows služby pro zpracování naskenovaných dat.

Dalším krokem pro realizaci tohoto projektu bylo naprogramovat samotnou Windows službu a propojit jí s již vytvořenou knihovnou pro komunikaci s laserovým skenerem.

4.1 Windows služba

Windows služba neboli anglicky service [11], je druh aplikace, která běží na pozadí systému. Tento druh aplikace nemá žádné grafické rozhraní a není třeba pro její chod žádného uživatele k obsluze. Jsou služby, jenž se spustí automaticky po startu počítače, a jiné které se spustí až na vyžádaní například uživatele. Služby jsou často používané na serverech nebo právě na PLC zařízeních. Zkompilovaná služba vystupuje ve formátu exe souboru, který je třeba zaregistrovat do systému, nelze jej přímo spustit jako jiné exe soubory. Po přímém spuštění je většinou uživatel upozorněn, že se jedná o službu a nelze ji spustit přímo, ale musí jí nejdříve zaregistrovat.

Tento druh aplikace je zkonstruován tak, aby mu byl systém schopný předávat příkazy jako jsou „Start“, „Stop“, „Pause“ a „Continue“. Hlavním rozdílem oproti tvorby běžné aplikace, která obsahuje uživatelské rozhraní je, že ve službě neběží hlavní vlákno, pouze obsluhuje příchozí příkazy jako jsou spuštění služby, zastavení služby a jiné. Je už jen na vývojáři, jak službu naprogramuje, aby reagovala na tyto příkazy.

Psání Windows služeb je přizpůsobeny Microsoft .NET Framework, a ve spolupráci s programovacím jazykem C# ve vývojovém prostředí Microsoft Visual Studio je vývoj služby velmi pohodlný. Při založení nového projekt ve Visual Studiu, lze vybrat přímo možnost Windows Service. Po založení projektu se vytvoří dva následující hlavní soubory:

- Program - v této třídě se definuje chování služby během spuštění, dá se v ní nastavit různé parametry s kterými bude služba spouštěná, například aby bylo možné službu spouštět v rámci ladění a testovat přímo z vývojového prostředí. Dále se tam nastavuje instalace a parametry s jakými bude služba zaregistrována v systému, jako je název, složka v které je služba umístěna a pro jaké uživatele bude přístupná.
- Service - v této třídě se implementuje, jak se má služba chovat a jakou funkcionalitu má vykonávat. Komunikaci s touto třídou zajišťuje třída Program při instalaci a spuštění služby.

4.2 Registrace služby

Během registrace služby se primárně definují tyto její nastavení:

- Jméno služby - jedná se o unikátní název služby, měl by se zadávat bez diakritiky a jiných problémových znaků.

- Zobrazované jméno služby - informativní jméno služby, zobrazené v systému. Může obsahovat diakritiku.
- Popis služby - jedná se o popis, uvádějící činnost dané služby, zobrazený například ve vlastnostech služby.
- Účet pod kterým se spouští - službu je nutné spustit pod určitým účtem, a proto je nutné jej nastavit.
- Způsob spuštění - služba se dá spustit buď automaticky po startu systému, manuálně na vyžádání uživatele, nebo se dá také její spuštění dočasně zakázat.

Samotná registrace se dá provést více způsoby, v našem případě byla vytvořená třída obsluhující tuto registraci pojmenována „ProjectInstaller“. V této třídě je nutné nastavit všechny vstupní parametry popsané výše. Následně se přistoupí k registraci zkompilované služby pomocí příkazového řádku, a to tak že se musíme dostat do složky v které jsou soubory služby uložené. Příkaz pro registraci služby je v tomto formátu: Umístění služby\Název služby.exe -install. Parametrem „install“ se zajistí, že se tato služba nainstaluje, což je zajištěno v třídě Program popsané výše. Nainstalovaná služba se následně dá spustit buď z příkazového řádku, nebo ve správě počítače v sekci služby. V tomto případě byl vytvořený dávkový soubor pro zjednodušení instalace a odinstalace vytvořené služby, pomocí pouhého spuštění tohoto dávkového souboru. Služba byla nastavená až se spouští automaticky po startu systému. V pozdější fázi vývoje projektu, což znamená během závěrečného testování, byl vytvořen instalátor, který velmi usnadňuje registraci a instalaci této služby. Po spuštění instalátoru, se pouze vybere místo kde chceme až se služba nainstaluje a dále se již instalátor postará o rozbalení všech potřebných souborů, včetně konfiguračních souborů.

4.3 Princip funkce vytvořené Windows služby

Služba vytvořená v této práci byla navržena tak, aby byla schopná pracovat v nepřetržitém provozu, obsluhovat skener, vyhodnocovat naskenovaná data a posílat výsledky do koncového zařízení. Je navržená tak aby jakoukoliv chybu, která může nastat během skenování, vyhodnocování nebo komunikaci s připojenými zařízeními dokázala sama vyřešit bez nutnosti jejího restartování nebo jiného zásahu.

Součásti služby jsou tři konfigurační soubory. Z toho jeden obsahuje parametry potřebné k připojení a nastavení skeneru a další parametry potřebné k výpočtu, viz. přiložený zdrojový kód.

```
<?xml version="1.0" encoding="utf-8"?>
  <Sicks>
    <Sick Name="Sick400" IPAdress="127.0.0.1" Port="6666" Rotation="89"
      EncoderStep="0,5" MinObjectCount="20" ZTolerance="20"
```

```
        CrateTolerance="40" NumberOfObjects="200" DiffTolerance="10"
        LengthOfScanningArea="1190" FilterFromLocation="30" PaletteZ="2070"
         MultipleNumberOfObjects="1" FilterHandle= "false" AngleTolerance="
        5">
      <sickDetectArea Name="Zone" Y="-300" Z="2400" RectWidth="650"
        RectHeight="1500" />
      </Sick>
    </Sicks>
```

Výpis 2: Struktura XML souboru pro připojení k laserovému skeneru a s parametry pro výpočty

Druhý konfigurační soubor Obsahuje všechny potřebné parametry k připojení na OPC server a zprovoznění komunikace s PLC. V posledním souboru jsou parametry pro přepravní nádoby, které jsou na daném místě skenované. Všechny tyto konfigurační soubory jsou ve formátu XML.

Po startu služby se načtou pomoci XML parsování všechny parametry z těchto souborů. Následně se vytvoří připojení k OPC serveru a nastaví se na něm všechny OPC položky. Následně dojde k vytvoření spojení z laserovým skenerem a provedou se všechny počáteční nastavení, které potřebujeme na zařízení nastavit jako je skenovací frekvence a úhlové rozlišení skeneru, které určuje z jakou rychlostí a jak přesné budou chodit naměřené hodnoty ze skeneru. Následně je třeba na zařízení nastavit, jak chceme, aby probíhalo měření. V tomto případě bylo nastaveno, že jednotlivé skeny budou spouštěny hardwarovou spouští, což znamená že jeden sken se vykoná s každým pohybem manipulátoru, na kterém je zařízení umístěno. Informace o pohybu manipulátoru je přiváděná do zařízení pomocí vstupů určených pro toto měření, a to jsou konkrétně vstup jedna a tři. Tento vstup jako zdroj spouštěče je třeba opět nastavit zasláním příslušné zprávy na skener. Posledním parametrem zaslaným na skener je nastavení enkodéru, který slouží k přepočítávání signálu zaslaného na vstupy popsané výše. Tím je počáteční nastavení skeneru dokončeno.

Na konec se spustí pracovní vlákno služby, které neustále kontroluje, zdali nedošlo k odpojení od OPC serveru nebo od skenovacího zařízení. V případě že došlo ke ztrátě komunikace tak se služba pokouší o obnovení spojení. Dále se již čeká, než přijde nějaký pokyn z OPC serveru, který se bude potom zpracovávat.

4.4 Logování služby

Další nezbytnou součástí Windows služby je sledovat její chování, vidět, jak služba pracuje, zdali pracuje správně a hlavně dojde-li k nějaké chybě. Je velmi důležité zjistit kde a proč se daná chyba vyskytla. Všechny tyto vlastnosti můžeme vidět díky tzv. logování [12]. V rámci logování se vytváří tzv. log, čímž je označovaný záznam nějaké činnosti, nebo také přímo soubor do nějž je logování zapisované. tento soubor má takřka vždy koncovku za názvem .log a je většinou ve formátu textového souboru. Logování lze nastavit a rozdělit na několik úrovní, a to tak že se nastaví jak a kolik detailních informací se do logů bude ukládat.

Pro logování této služby bylo využito nástroje pro logování, který je dostupný v rámci interního vývoje firmy, v níž byla tato práce vytvářená. Aby bylo možné použít logování tak je pouze potřeba v aplikaci zaregistrovat příslušnou knihovnu, která se jmenuje „CLog“. Inicializace se provede spuštěním metody „Create (název služby)“ a do parametru se zadá název služby, podle které se bude zobrazovat od které aplikace daný log pochází. Potom se již pouze použije metody pro zápis těchto konkrétních událostí:

- Error - jedná se o výpis kritických chyb vyvolaných spuštěním nějaké výjimky, které většinou neumožňují pokračovat v procesu.
- Warning - v tomto případě by se měly vypisovat varování, které signalizují nějaký neočekávaný stav, který neumožňuje běžný běh procesu, avšak daný proces se může opět obnovit.
- Info - těmito logy se vypisují nejčastěji nějaké informace, které by mohly být pro uživatele důležité. Jedná se o informace o změně procesu, nebo nějakého stavu.
- Trace - jedná se o tzv. trasovací zprávy, jenž jsou obvykle užitečné během ladění aplikace.

Během vytváření služby je velmi důležité dodržet toto rozdělení logovacích událostí a nejlépe použít všechny čtyři druhy, což bylo v této službě splněno. Ukázka zápisu těchto logů je zobrazená v přiloženém kódu. Do prvního parametru pro příslušný log se vždy zapíše, o jakou třídu, metodu případně místo jde, což nám pomůže později určit v jakém místě se daný log provádí. V druhém parametru se zadává již příslušná zpráva, kterou chceme sdělit.

```
//registrace knihovny pro logovani
CLog.Create (Servicename);
//aktivovani logovani
CLogRtsEx.Active = true;

CLog.Error.Report("RtsDyAut.DetectingCrates.LoadCrateConfig", ex.
    Message);
CLog.Warning.Report("RtsDyAut.DetectingCrates.LoadConfig", "Sick is not
     exist. ");
CLog.Trace("DetectingCrates.SetReady", "aplicationReady = true");
CLog.Info.Report ("ServiceState", "Service has started");
```

Výpis 3: Zápis logování ve vytvořené službě

5 Tvorba algoritmu pro efektivní nalezení bodů přepravek z naměřených dat

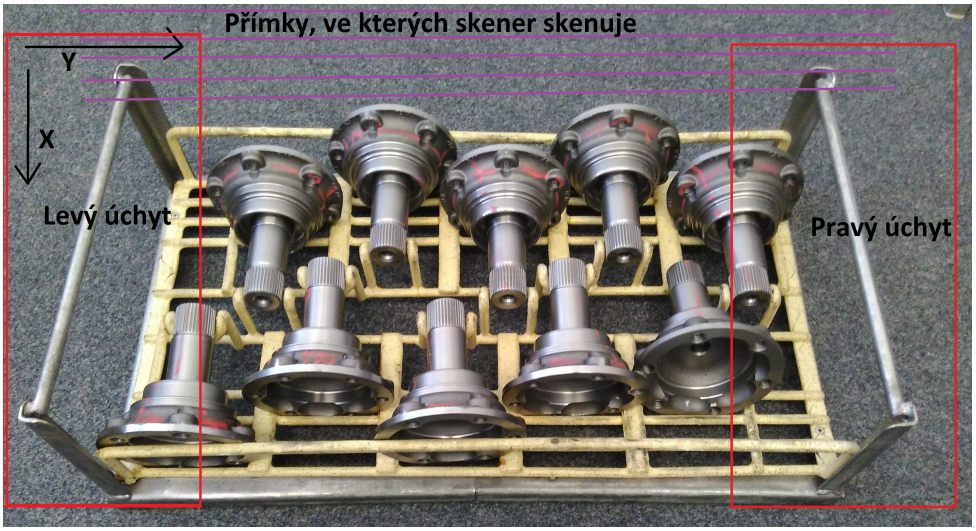
Po vytvoření služby a korektním zprovoznění komunikace se skenerem se následně přistoupilo k vytvoření samotného algoritmu pro zpracování naskenovaných dat, které přišly ze skeneru. Algoritmus byl přizpůsoben přímo typu přepravních nádob. Jak je vidět na následujícím obrázku 8, tak je na nakreslených šipkách zobrazený směr skenování.

Skener je vždy postavený nad skenovanou oblastí a skenuje pouze v jedné přesné přímce, jak je zobrazené na obrázku pomocí fialových čar. V reálném prostředí lze vždy vidět, když skener skenuje, laserovou čáru v místě, kde má skener momentálně dosah. Jak si lze z obrázku povšimnout postavení přepravní nádoby vůči skenované přímce, tak toto postavení bude vždycky v tomto směru, ne opačně. Díky tomu máme jistotu, že v každém skenu, který bude proveden již přímo nad přepravkou, tak bude zachycený alespoň jeden úchyt přepravky, avšak ve většině skenů se budou nacházet body odražené z obou úchytů. Tyto úchyty jsou zobrazené na obrázku v červených obdélnících. Všechny skenované přepravní nádoby jsou víceméně stejné konstrukce, liší se pouze v rozměrech. A právě kvůli těmto vlastnostem bylo přistoupeno k vytvoření algoritmu, jenž byl navržen v návrhu této práce. Tento algoritmus je založený na principu, že se pokouší z každého skenu nalézt vrcholy přepravní nádoby, čímž jsou již popsané úchyty.

5.1 Zpracování dat z jednotlivých skenů

Komunikace se skenerem probíhá pomocí telegramů, které již byly popsané v kapitole zabývající se vytvořením knihovny pro tuto komunikaci. Jak již bylo nastíněno v návrhu, tak celý princip skenování v celém skenovacím prostoru probíhá tak, že skener je umístěn na konstrukci s manipulátorem a pohyb tohoto manipulátoru je řízen krokovým motorkem. Z tohoto motorku je přiváděn signál na skener, jenž přepočítává otáčky na milimetry, znamenající délku posunu vpřed. V tomto případě bylo nastaveno, že signál na vstup skeneru přes tzv. enkodér přijde každého půl milimetrů ve formě logické jedničky. Po příchodu tohoto signálu se provede na skeneru právě jeden sken zachycující konkrétní přímku nacházející se kolmo pod skenerem ve skenovací oblasti. Tímto nastavením skenování s krokem půl milimetru byla dosažená velmi vysoká přesnost. Aby bylo dosaženo tohoto druhu skenování, jenž je řízeno právě tímto příchozím signálem na enkodér z krokového motorku, tak je nutné na skener odeslat sérii příkazu, které jsou zobrazené v následujícím kódu.

```
_sick.SetTriggeringControlledSettings ("sWN IObase 01");
_sick.SetDigitalInputSettings ("sWN IOpins 0000 0000 01 0000 0000 01 0000
    0000 01 0000 0000 01");
_sick.SickSetGate ("sWN IOgcfg 01 08 07D0 0000 0014 0032 00 0000 0000 0000
    0000");
_sick.SetEncoder ("sWN IOencm 01");
```



Obrázek 8: Skenovaná přepravní nádoba

Výpis 4: Nastavení skenování řízené hardwarovou spouští

Toto nastavení bylo již částečně zmíněno v kapitole 4.3. avšak tady bude popsané podrobněji. Prvním příkazem se nastaví, že skenování bude řízeno spouštěčem, tzv. „triggerem“, a „trigger“ bude řízený vzdáleností. Druhým příkazem se nastaví jednotlivé vstupy. Skener má čtyři vstupy, a tak se pro každý vstup nastaví zpoždění v milisekundách nebo v milimetrech po kterých se má skenování spustit. V našem případě to je všude nula, jelikož chceme až skener provede sken okamžitě.

V třetím příkazu se provede nastavení, že je aktivní vstup jedna a tři, a právě na tyto vstupy bude chodit signál z krokového motorku, kterým se bude spouštět skenování. Následující parametry v tomto příkazu zde nebudou popsány, jelikož vše je podrobně popsané v dokumentaci a pro vysvětlení principu funkce nejsou důležité. Posledním příkazem se definuje typ enkodéru, jenž převádí signál přiváděny na vstup z krokového motorku. Po této sérii nastavení se skenování spouští tímto způsobem.

```
_sick.StartTriggeredMeasurement ("sMN mLRreqtrigdata " + (int)typeOfMeasure)
    ;
_sick.StopMeasurement ("sMN mLRstopdata");
```

Výpis 5: Spuštění skenování řízené hardwarovou spouští

Prvním příkazem se provede spuštění skenování, podle parametrů, které byly nastaveny v předchozím kódu. Od této doby bude skener posílat data z naskenovanými hodnotami, s každým příchozím signálem na vstup, jak bylo popsané výše. Celý skenovací proces je tedy ukončen až do poslání druhého příkazu, po jeho provedení již ze skeneru nebudou chodit žádné naskenovaná data.

Tím konečně můžeme přistoupit k popisu zpracování jednotlivých skenů, kde každý sken obsahuje mezi jinými tyto důležité parametry:

- Format - formát naskenovaných dat. Skener může vracet dva formáty dat. V prvním případě vrací pouze naměřené vzdálenosti jednotlivých bodů, v druhém případě vrací se vzdálenostmi také odrazivost daného bodu. V tomto případě byl použit první formát.
- StartingAngle - úhel od kterého skener začal měřit.
- AngularStepWidth - krok mezi jednotlivými naměřenými vzdálenostmi udávaný v stupních.
- NumberMeasuredValues - počet naměřených hodnot. Tato hodnota je vždy konstantní a je závislá na nastavení s jakým úhlovým krokem má skener měřit. V tomto případě byl skener nastavený s takovou přesností, aby vracel co největší počet naskenovaných hodnot v jednom měření, což je sedm set.
- ScanningFrequency - skenovací frekvence v jednotkách Hertz, udávající rychlost skenování skeneru.
- Distance_n - následuje balík dat dlouhý podle parametru určujícího počet naměřených dat s jednotlivými vzdálenostmi.
- EncoderPosition - pozice enkodéru. Tato pozice udává na jaké pozici se momentálně skener nachází, tuto pozici přičítá podle počtu impulsů, které přijdou na vstup.

Po přečtení celého balíku dat se spustí funkce, která má za úkol přepočítat naskenované vzdálenosti na 3D souřadnice. K tomuto přepočtu jsou důležité právě příchozí parametry pro krok mezi jednotlivými body a počáteční úhel od kterého byl sken proveden.

```
double angle = StartingAngle
for (int i~= 8; i~< data.Count - 9; i++) {
   UInt16 distance = (UInt16)data[i];

   Distance = (double)distance;
   pointY = Distance * Math.Sin (angle * (Math.PI / 180));
   pointZ = Distance * Math.Cos (angle * (Math.PI / 180));

   _point3D = new Point3D((float)_encoderPosition, (float)pointY, (float)
      pointZ);

   angle += AngularStepWidth;
}
```

Výpis 6: Přepočet naměřených vzdáleností na 3D souřadnice

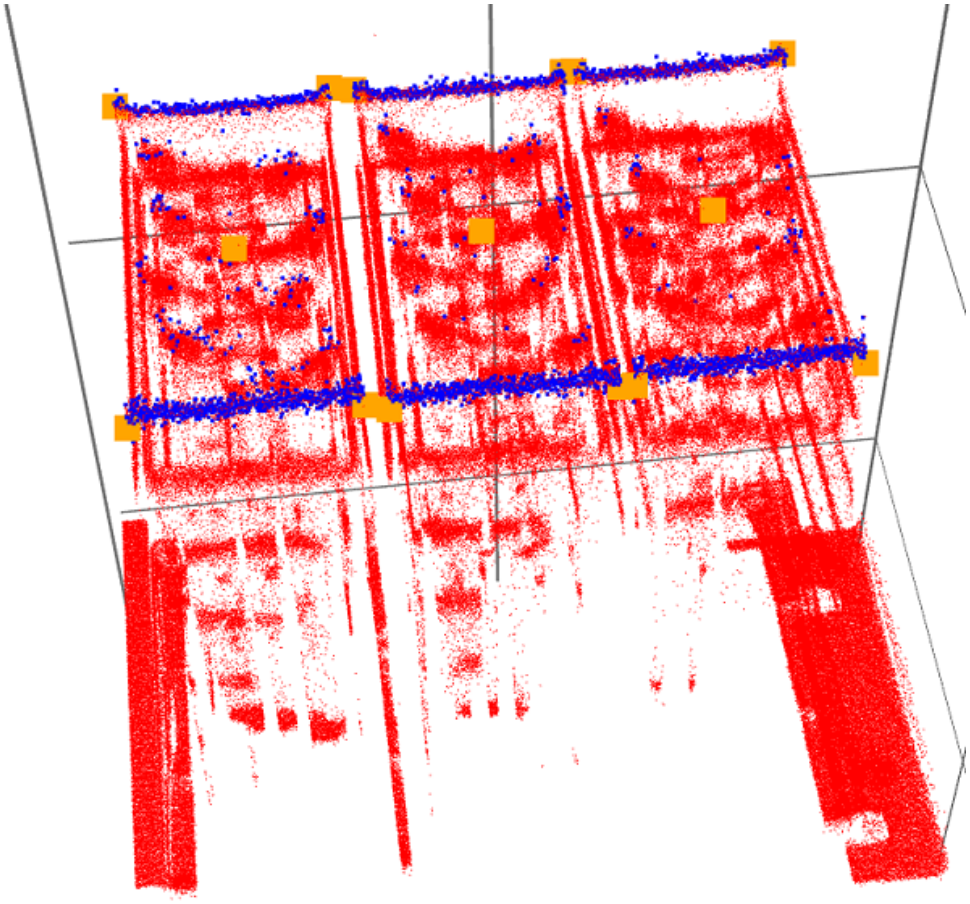
Jak je v následujícím kódu vidět, souřadnice v ose X je získávána z polohy enkodéru a je to vzdálenost kterou urazil skener během skenování. Tato hodnota je pro všechny vzdálenosti v jednom skenu stejná. Souřadnice v ose Y je vzdálenost do strany od středu skeneru a souřadnice v ose Z je přepočtená vzdálenost od skeneru pro tento konkrétní bod v ose Y. Tímto převodem se přepočítají všechny naměřené hodnoty z jednoho skenu, v našem případě jich je sedm set, a uloží se do Listu s 3D body.

Následně tyto body jsou zpracovaný stejným způsobem, jak bylo již podrobně popsáno v návrhu. Algoritmus se snaží najít již zmiňované úchyty přepravek a v případě jejich nalezení vytvoří objekt obsahující všechny body mezi těmito vrcholy a spočítá jejich parametry. Tento algoritmus se provede následně pro všechny skeny, které nám přijdou v průběhu jednoho skenovacího procesu.

5.2 Zpracování dat ze všech skenů

Předchozí zpracování jednotlivých příchozích skenů probíhalo za běhu během skenování. Následující zpracování bylo spuštěno až po dokončení skenovacího procesu. Zpracování probíhalo opět jak již bylo znázorněno v návrhu, a to tak, že se prošly jednotlivé objekty představující fragmenty přepravních nádob. Dále se odfiltrovaly podle nastavených parametrů objekty, které nepocházely z přepravních nádob, ale například z mezer mezi jednotlivými sloupci, nebo z prostoru v kterém žádná nádoba nebyla. Po vyfiltrování všech nežádoucích objektů se z nalezených objektů poskládají jednotlivé přepravní nádoby. Jak vypadá mračno naskenovaných bodů ze skenovacího prostoru je zobrazené na následujícím obrázku 3D modelu 9.

Na obrázku je dobře vidět nalezené přepravky. Jedná se o horní řadu z každého ze tří sloupců, v jednotlivých sloupcích bylo na sobě až osm přepravních nádob. Zobrazené červené body jsou mračno bodů, které vzniklo z přepočítání jednotlivých vzdálenosti naměřených skenerem v každém skenu, přepočteném na 3D body. Modrými tečkami jsou již označené ty body, jež byly nalezeny vyhledávacím algoritmem a z nichž se skládají úchyty přepravních nádob. Oranžovými čtverci jsou označené rohy jednotlivých přepravek a jejich středy.



Obrázek 9: 3D model naskenovaného prostoru pod skenerem s nalezenými přepravními nádobami

6 Tvorba algoritmu pro vypočítání souřadnic pro jednotlivé přepravní nádoby

Po nalezení jednotlivých přepravních nádob bylo následně potřeba vypočítat jejich průsečíky, rozměry, a nakonec souřadnice jejich středů. Při tvorbě tohoto algoritmu se vycházelo z návrhu, avšak během testovacího procesu se provedly některé změny, zejména se upravil výpočet úhlu natočení přepravky. Postup výpočtu je zobrazený v následujícím výpisu zdrojového kódu.

```
        for (int i~= 0; i~< crateList.Count; i++) {
           {
              var minZ = CalculateMinZ (crateList[i].ListObject);
              if (crateList[i].ListObject.Count > _numberOfObjects) {
                 double leftY = crateList[i].ListObject[0].LeftPoint.Y;
                 double rightY = crateList[i].ListObject[0].RightPoint.Y;
                 double minX = crateList[i].ListObject[0].LeftPoint.X;
                 double maxX = crateList[i].ListObject[crateList[i].ListObject.
                     Count - 1].LeftPoint.X;

                 for (int j = 0; j < crateList[i].ListObject.Count; j++) {
                    if (crateList[i].ListObject[j].LeftPoint.Y < leftY &&
                        crateList[i].ListObject[j].LeftPoint.Z <= Math.Abs (minZ
                         + _crateTolerance)) {
                       leftY = crateList[i].ListObject[j].LeftPoint.Y;
                    }
                    if (crateList[i].ListObject[j].RightPoint.Y > rightY &&
                        crateList[i].ListObject[j].RightPoint.Z <= Math.Abs (
                        minZ + _crateTolerance)) {
                       rightY = crateList[i].ListObject[j].RightPoint.Y;
                    }
                 }
                 var tmpWidth = Math.Abs (leftY - rightY);
                 CalculatedCrateList.Add (CalculateCrate (crateList[i],
                     tmpWidth, minX, maxX, leftY, rightY, count));
                 count++;

              }
           }
        }
```

Výpis 7: Výpočet parametrů pro jednotlivé přepravní nádoby

Z tohoto kódu je patrné, že se postupně projdou všechny nalezené přepravní nádoby. Následně se najde nejmenší hodnota v ose Z pro všechny objekty, s kterých se aktuální nádoba skládá. Je to důležité, jelikož podle tohoto bodu se nastaví výsledné souřadnice v ose Z, čímž se definuje vzdálenost přepravky od skeneru. Může se totiž stát, že nějaký díl může být špatně umístěn, a může vyčnívat. Kdyby nebyla zadaná vzdálenost pro tuto součástku od skeneru, ale vzdálenost úchytů nádoby, tak by mohlo dojít k zablokování manipulátoru a naražení do součástky vyčnívající z přepravní nádoby. Dalším krokem je nalezení minimální a maximální souřadnice v ose X a Y. Dále se ještě spočítá momentální šířka objektu a všechny vypočtené parametry se pošlou do další funkce.

```
//Calculation of the angles from both sides of the box shall be made
a~= Math.Abs (nearestLeftPoint.Y - nearestRightPoint.Y);
b = Math.Abs (nearestLeftPoint.X - maxX);
angleNearest = (Math.Asin (b / a)) * (180 / Math.PI);
a2 = Math.Abs (farthestLeftPoint.Y - farthestRightPoint.Y);
b2 = Math.Abs (farthestLeftPoint.X - minX);
angleFarthest = (Math.Asin (b2 / a2)) * (180 / Math.PI);
//The larger angle is stored
if (angleNearest >= angleFarthest) {
   crateFrom3DPoints.Angle = angleNearest;
}
if (angleNearest < angleFarthest) {
   crateFrom3DPoints.Angle = angleFarthest;
}
//The coordinates of the crate is calculated
width = Math.Abs (leftY - rightY);
length = Math.Abs (minX - maxX);
b = b / 2;
if (b > _crateTolerance) {
   tmpLength = length;
} else {
   tmpLength = length - b;
}
crateFrom3DPoints.Width = tmpWidth;
crateFrom3DPoints.Length = tmpLength;
diagonalY = leftY + width / 2;
diagonalX = minX + tmpLength / 2;
```

Výpis 8: Výpočet souřadnic úhlu natočení a rozměrů přepravní nádoby

V této funkci se provede nejdříve výpočet úhlu natočení z obou stran. Výpočet je založený na tom, že se najde poslední sken, v kterém jsou zachycené oba úchyty. Následně se projdou skeny, které obsahují již jen jeden úchyt až po poslední a z toho se vypočítá strana b. Strana a je z rozdílu mezi minimálním a maximálním bodem v ose Y, který reprezentuje šířku objektu. Následně se pomocí sinové věty provede výpočet úhlu natočení přepravní nádoby. Stejný postup se provede také pro druhý konec přepravky, z důvodu větší přesnosti. Úhel natočení byl velmi důležitý, jelikož manipulátor měl nastavenou určitou mez, při jakém úhlu mohl přepravní nádobu vzít a přemístit na dopravníkový pás.

Algoritmus tedy vybere větší z naměřených úhlu z obou stran a dále vypočítá rozměry nádoby. Od těchto rozměrů se odpočítá délka, která vznikla z obou stran natočením nádoby, a to z toho důvodu, aby naměřené rozměry byly co nejpřesnější. Tyto rozměry jsou pak použité v závěrečné kontrole, než se zapíšou výsledky, při které se porovnávají rozměry nalezených přepravek s parametry z konfiguračního souboru. Tyto parametry udávají rozměry přepravních nádob, které se v daném místě můžou vyskytovat. Nakonec se vypočítají průsečíky a tím se stanoví souřadnice středů nádoby. Na tyto souřadnice bude následně naváděn manipulátor, aby mohl přepravky uchytit a přemístit.

7 Vytvoření komunikace s OPC serverem

Jak již bylo v návrhu této práce zmíněno, tak komunikace mezi PLC, manipulátorem a zde popisovanou službou pro obsluhu laserového skeneru, bude probíhat za použití OPC protokolu [13].

7.1 Popis OPC protokolu

Jedná se o sadu protokolů vytvořených v roce 1996 sdružením OPC Foundation, určenou zejména pro průmyslovou automatizací. Definuje problematiku komunikačního rozhraní při řízení a monitorování technologických procesů. Protokol byl založený zejména kvůli zkoušení komunikace mezi řídícími zařízeními a kancelářských aplikací v oblasti řízení procesů.

Tato sada protokolů definuje standardní objekty, vlastnosti a metody pro servery poskytující informace v reálném čase. Nejčastější použití těchto protokolů je právě u PLC automatů, distribuovaných řídících systému, inteligentních snímačů a jiných. Tato specifikace byla založená na OLE, COM, DCOM technologií vyvinutých společností Microsoft pro rodinu operačních systému Windows. Komunikace pomocí OPC protokolu je založená na architektuře klient-server, kde:

- Server poskytuje všechny data a zajišťuje komunikaci ze všemi fyzickými zařízeními.
- Klient komunikuje ze serverem, jenž provádí všechny požadované operace.

OPC protokol se dále dělí na tyto dvě hlavní specifikace:

- OPC DA [15] - tato specifikace je zaměřená zejména pro sběr dat ze zařízení jako jsou PLC automaty v reálném čase. Je zde kladen důraz na průběžnou výměnu dat. Zabývá se pouze daty reálného času, pro histiorická data, nebo události a normy je rozšíření tohoto protokolu OPC Historical Data Access a OPC Alarms and Events. Data z tohoto protokolu se skládají z těchto tří atributů:
  - Hodnota
  - Kvalita hodnoty
  - Časové razítko

  Dle této specifikace je nutné vrátit OPC klientovi vždy všechny tři tyto atributy. Pokud zdroj dat není schopen poskytnout časové razítko, tak ho OPC DA server musí vytvořit sám
- OPC UA - jedná se o nástupce předchozí specifikace, tím pádem je založená na stejném principu, avšak poskytuje několik nových vylepšení jako je nezávislost na platformě a komplexnější možnosti informačního modelování.

7.2 OPC server

OPC server [14] se používá zejména z toho důvodu, že každý hardware má nějaký svůj ovladač, prostřednictvím kterého komunikuje s koncovým zařízením a tím vzniká nezměrný počet problémů. Mezi tyto problémy patří zejména to, že aplikace musí obsahovat ovladač pro každý hardware ke kterému je připojena a tento ovladač může být časem změněn, nebo může dokonce nastat nějaký konflikt mezi jednotlivými ovladači. Těmto všem problémům právě předchází OPC specifikace, která poskytuje jednotné rozhraní pro výměnu dat mezi OPC klienty, jenž jsou koncovými zařízeními a OPC serverem, který poskytuje všechny data. Všechny aplikace napojené na toto rozhraní nejsou dále závislé na použití konkrétních ovladačů pro jednotlivá zařízení.

Díky těmto výhodám, které byly popsané, je komunikace v této práci založena také na OPC protokolu. OPC server byl umístěný na PLC panelu. Připojení na tento server se provedlo pomocí vytvoření OPC klienta přímo ve službě. Pro vytvoření klienta byla použitá knihovna, která je dostupná v rámci interního vývoje firmy, v níž byla tato práce vytvořená. Jak vypadá vytvoření OPC klienta je zobrazeno v následujícím kódu.

```
OpcServer = new OpcServerCln ();
OpcServer.Connect (OpcName, OpcHost);
OpcServer.AddGroup ("Groups", true, 100);
OpcGrp = OpcServer["Groups"];
OpcGrp.OnDataChanged += OpcGrp_OnDataChanged;

//Add OPC items and Groups to OPC klient
foreach (string tag in Tags) {
   OpcTags.Add (tag);
}
_sickOpc.AddGroup (GroupName, OpcTags);
_sickOpc.SetOpcValues ();
```

Výpis 9: Vytvoření OPC klienta a komunikace se serverem

Na počátku je nutné vytvořit novou instanci této knihovny. Následně se zavolá metoda „Connect“, do které se jako parametry zadají název OPC serveru a IP adresu na které server běží. Následně se na klienta vytvoří OPC skupina, kde se zadá její název a časový údaj v milisekundách, po kterých se kontroluje změna na serveru. Dále se nastaví událost, která bude kontrolovat veškeré změny, které nastanou na OPC položkách na serveru. V druhé části kódu je ukázán způsob, jak se přidávají jednotlivé skupiny z jejích položkami do klienta. Následně se zavolá metoda, jenž nastaví hodnoty pro všechny zadané OPC položky. Seznam OPC položek pomocí kterých komunikuje vytvořená služba z PLC zařízením je zobrazený pomocí následující tabulky 2 .

Tabulka 2: Tabulka OPC položek

Skupina	**Podskupina**	**Prvek**
OpcOutput		axisXPositionOnStart
		start
OpcInput		aplicationReady
		scanning
		resultsValid
		error
OpcInput	boxPosition[0]	centerX
		centerY
		centerZ
		angleZ
		dimensionX
		dimensionY
		isPlaced
OpcInput	boxPosition[1]	centerX
		centerY
		centerZ
		angleZ
		dimensionX
		dimensionY
		isPlaced
OpcInput	boxPosition[2]	centerX
		centerY
		centerZ
		angleZ
		dimensionX
		dimensionY
		isPlaced

Položky ze skupiny „OpcOutput“ jsou položky, které mění PLC počítač, a čímž dochází k nastavení souřadnice v ose X, což je aktuální poloha skeneru, před skenováním. Druhá položka v této skupině spouští a ukončuje skenovací proces.

Položky ze skupiny „OpcInput“ jsou naopak měněné pouze touto aplikací, což znamená že se do těchto položek zapisují výsledky vyhledávacích algoritmů a stavy aplikace. Do prvních čtyř položek se zapisuje postupně stav, že je služba připravená na spuštění skenování, oznámení že aplikace momentálně je v procesu skenování a zpracovává data, dále že jsou naměřené výsledky v pořádku a v poslední položce se zapisuje, že došlo k nějaké chybě. Následují OPC položky pro jednotlivé přepravní nádoby. Jelikož je možné během jednoho skenu dle zadání nalézt maximálně tři přepravní nádoby, tak jsou i tyto položky rozdělené do tří skupin. V každé skupině se zapíšou výsledky vyhledávacích algoritmů, což jsou souřadnice, úhel natočení a rozměry v ose X a Y pro přepravky. Poslední položkou se nastavovalo, že je daná přepravní nádoba skutečně nalezená, a že může být manipulátor na tyto souřadnice naváděn.

Po zprovoznění OPC klienta a načtení všech výše zmíněných položek, již byla vytvářená služba kompletně schopná být nasazena na PLC panel a mohlo se následně přistoupit ke zprovoznění a testování provozu.

8 Vytvoření 3D modelu pro zobrazení naměřených dat

Ještě, než bylo přistoupeno k testování vytvořené Windows služby přímo v reálném provozu, tak bylo potřeba vytvořit WPF aplikaci pro zobrazení naskenovaného mračna bodů z jednotlivých skenů. Hlavním požadavkem na tuto aplikaci bylo, aby se dal načíst soubor z jakéhokoliv skenovacího procesu a zobrazit, jak vypadá naskenovaný 3D model. Dále bylo důležité, vytvořit uživatelské rozhraní tak, aby se dal vytvořený model přibližovat a zobrazit z libovolného úhlu.

8.1 Popis použitých komponent pro zobrazení 3D modelu

Pro vytvoření 3D modelu z naměřených dat byla použitá komponenta Helix Toolkit. Tato komponenta je volně dostupná v licenci open-source a je postavená na .NET Frameworku. Je uzpůsobena pro použití ve WPF aplikacích. Tato komponenta poskytuje knihovny a metody pro vytvoření 3D modelu a jeho vizualizace. Dá se jednoduše přidat do projektu ve Visual Studiu pomocí tzv. NuGet balíčku.

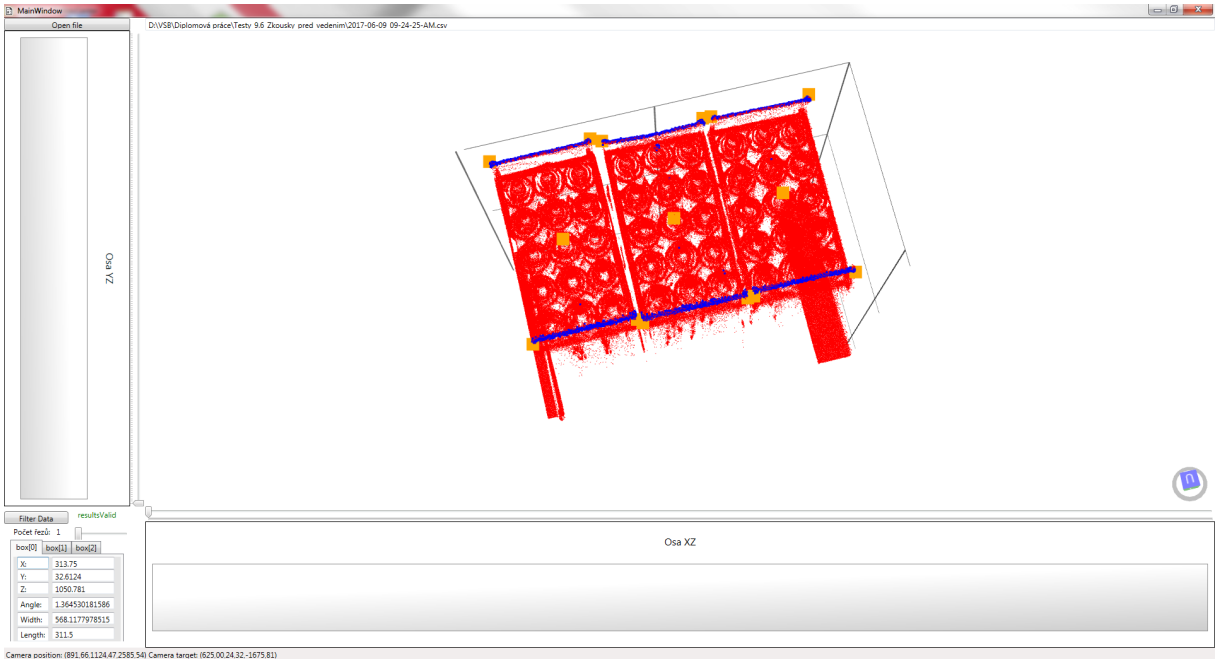
Při vytváření aplikace se postupovalo těmito kroky. Nejdříve se vytvořil WPF projekt. Dále se za použití návrhového vzoru MVVM jenž již byl podrobně popsán v návrhu vytvoří propojení mezi uživatelským rozhraním a funkcemi aplikace. Dalším krokem bylo přidat do aplikace výše popsanou komponentu. Z této komponenty se použilo v uživatelském rozhraní HelixViewport3D, čím se zajistilo vykreslení celého 3D modelu. V této komponentě je nutné nastavit zdroj zobrazovaných 3D bodů. Tento zdroj se nastavil pomocí „data bindingu“, jak pro všechny body naskenované skenerem, tak pro konkrétní body nalezené vyhledávacím algoritmem pro vyhledávání samotných přepravek. Tyto zdroje dat se následně nastavily ve ViewModelu.

8.2 Popis problematiky zpracování naměřených dat

Samotné naskenovaná data se získávaly z jednotlivých souborů, které byly ve formátu CSV dokumentu. Tyto soubory jsou vytvářené vždy po dokončení skenovacího cyklu, kdy služba zapíše všechny naskenovaná data převedená na 3D body do souboru. Požadavkem bylo, aby se zachovávalo vždy posledních sto skenů provedených na místě. Po spuštění aplikace má uživatel možnost si vybrat jakýkoliv soubor s naskenovanými daty. Po vybrání souboru dojde tedy k načtení všech dat. Tyto data jsou uložená ve formě X, Y, Z sloupců pro každý řádek. Po přečtení každého řádku se vytvoří 3D bod a ten se uloží do Listu těchto bodů. Když jsou načtená data z celého souboru tak se tyto data nastaví jako zdroj bodů pro komponentu zobrazující 3D model. Vzhledem k datové náročnosti byl přidán do aplikace posuvník z možností volby dělitele počtů skenů, které se budou zobrazovat. Jelikož během skenování se prováděl každý jeden sken, co půl milimetru, tak je naskenovaných dat opravdu hodně, a aplikace je nastavená, že zobrazuje primárně každý desátý sken. Díky tomu lze modelem svižně pohybovat a případně si ho zmenšovat nebo zvětšovat.

8.3 Popis vzhledu a funkcí výsledné aplikace

Tato aplikace byla dále rozšířena o možnost filtrování naskenovaných dat. Tímto filtrováním se spustí úplně stejné vyhledávací algoritmy, které se používají ve službě. Díky tomu si lze zobrazit úplně stejné výsledky a podrobně zobrazit jednotlivé přepravní nádoby které, byly v reálné situaci nalezené. Díky těmto funkcím byla tato aplikace velmi využívána při optimalizaci vyhledávacích algoritmů při zavádění do provozu.



Obrázek 10: WPF aplikace pro zobrazení 3D modelu naskenovaných dat

Jak je vidět na obrázku 10, kde je zobrazené uživatelské rozhraní aplikace, tak v levém dolním rohu, se v jednotlivých záložkách pro přepravní nádoby zobrazí výsledky vyhledávacího algoritmu. 3D model je v tomto případě nastavený tak, že zobrazuje každý sken, který byl poslán ze skeneru. Na výsledném modelu, lze zobrazit přesnost skenování, jelikož jsou jasně viditelné i samotné díly, které jsou umístěné v přepravních nádobách. Dále je vidět na modelu nalezené úchyty přepravních nádob zobrazené modrými tečkami a vyznačení jednotlivých přepravních nádob pomocí oranžových čtverců. Ze středů každé přepravní nádoby jsou pak vypočítávané výsledné souřadnice, které jsou zobrazené ve výsledcích.

9 Testování (Unit testy)

Součástí vývoje každé aplikace je nezbytné vytvořit testování. Testování se provádí zkoušením samotné aplikace v určitých situacích, tak i vytvořením automatických testů, které testují nejdůležitější součástí logiky samotné aplikace. V následující kapitole bude popsáno, jak probíhalo testování samotných aplikací během vývoje.

9.1 Počáteční vývoj a testování v kanceláři

Na samém počátku, během vytváření knihovny pro obsluhu laserového skeneru se nejdříve testovala funkčnost skeneru samotného. To probíhalo tak, že se laserový skener pověsil v kanceláři do výšky přibližně jednoho a půl metru. Následně se začala vyvíjet samotná knihovna a zkoušelo se jednotlivé příkazy, jak pro spouštění skenování, tak pro nastavení parametrů skenování. Zkoušelo se, jak skener vrací data při odlišném nastavení frekvence a kroku mezi jednotlivými body. Po vytvoření knihovny a nastavení skeneru pro nejlepší přesnost skenování, se přistoupilo ke skenování již samotných přepravních nádob. Pro zobrazení jednotlivých skenů byla vytvořená jednoduchá aplikace, s které následně vycházela aplikace pro zobrazení 3D modelu. Vyhledávací algoritmus byl vytvářeny od úplných základů, a to tak, že se nejdříve vytvořilo vyhledávání fragmentů přepravek z jednoho skenu. Poté bylo dalším krokem upravit algoritmus tak, aby byl schopný nalézt fragmenty přepravních nádob z více skenů najednou a sestavit již přibližný 3D model. Jelikož toto testování bylo prováděno v kancelářských podmínkách, protože se konstrukce s manipulátorem teprve vytvářela, tak skenování přepravek nebylo úplně přesné. Avšak na vytvoření základního vyhledávacího algoritmu to plně stačilo.

Během celého vývoje vyhledávacího algoritmu a obou aplikací byl použit iterativní vývoj, takže se začínalo od základních funkcí, a tyto funkce byly následně dále rozšiřovaný a byly přidávané nové a nové funkce. Při tvorbě aplikací se vycházelo z analýzy a návrhu, kde již byl navržen vyhledávací algoritmus a byly sepsány všechny požadavky.

Když byl vyhledávací algoritmus již dostatečně přesný, tak byla vytvořená již popisovaná Windows služba, do ni přidány všechny již vytvořené součástí, a byla přidána komunikace s OPC serverem. Tato komunikace byla odzkoušená pomocí vytvoření lokálního serveru, kde se nahrály všechny potřebné OPC položky. Po vytvoření této komunikace již byla služba testována během jednotlivých skenovacích procesů spouštěním pomocí změn v OPC položkách. Pro spouštění jednotlivých skenů, byla použitá dočasně softwarová spoušť, jelikož v konečné podobě bylo skenování spouštěné pohybem manipulátoru, jak již bylo popisováno.

9.2 Tvorba Unit testů

Další velmi důležitou součásti testování aplikací během vývoje, je vytvoření Unit testů. Unit testy[16] jsou tzv. jednotkové testy a většinou mají na starost automatické testování určité části kódu jako jsou metody vytvořených knihoven, nebo funkce vytvářející nějaké výpočty. U tohoto

typu testování se testuje třída a její metody, a to tak, že se testuje jedna metoda po druhé. Pro každou metodu se snažíme vytvořit několik testů, do kterých se zadávají různé vstupy, tak aby se dalo co nejlépe odhalit nějakou chybu. Jedná se o tzv. „whitebox“ testy, což znamená, že se píšou s tím, že programátor je obeznámen jak testovaný kód funguje.

Během vytváření Unit testů je důležité dodržet nezávislost jednotlivých testů na sobě. Během testování by měla být tendence izolovat testovanou jednotku od ostatních. Za tímto účelem izolovat jednotlivé jednotky od sebe se často používají pomocné objekty. Tyto objekty simulují předpokládaný kontext, ve kterém testována část kódu pracuje. Tento typ objektů se jmenuje tzv. „mock“ objekt a vytváření těchto objektů se nazývá tzv. „mockování“.

Unit testy by se také měly psát na základě návrhu, ne na základě implementace, což znamená, že se vytvářejí na základě očekávané funkčnosti. Díky tomu by se mělo snadněji odhalit případné chyby v kódu. Pro vytváření Unit testů jsou potřeba následující nástroje [17]:

- Testovací framework - jedná se o nezbytnou součást pro vytváření jednotkových testů. V tomto Frameworku lze vytvářet jednotlivé testy a také umožňuje označit nebo naopak vyloučit jednotlivé třídy a metody z testování. Dá se v něm také nastavit, které testy se spustí před testem, po testu nebo po celé či před celou třídou. Toto označování a nastavování se provádí pomocí datových anotací nad samotnými třídami a metodami. Poskytuje řadu porovnávacích funkcí, které porovnávají očekávané a skutečné výsledky. Mezi nejpoužívanější testovací Frameworky patří:
  - MsTest
  - NUnit
  - xUnit
- Test runner - tento nástroj se používá pro vyhledávaní a spuštění již vytvořených Unit testů. Nástroj poté vrací na základě výsledků porovnávacích funkcí tzv. „assertů“ stav zásobníku, kde jsou zobrazené a popsané jednotlivé chyby. Visual studio již obsahuje tento nástroj, a proto není nutné jej instalovat, avšak existuje mnoho nástrojů třetích stran, které mají rozšířené možnosti používání.
- Mockovací framework - tento nástroj není k samotné tvorbě Unit testů potřebný, avšak při kvalitním vytváření Unit testů je nezbytný, jelikož je jeho využití velmi široké. Tento Framework umožňuje realizovat různé metody a rozhraní a to tak, že jim nadefinuje jejich chování. Tento nástroj se používá zejména při testování nějaké metody, která využívá jinou závislost. Tato závislosti se pak tzv. „mockuje“, což znamená, že se nastaví její předpokládané chování na základě vstupu. Díky tomuto nástroji můžeme testovat metody, bez toho, aby byly ovlivněné jinými částmi aplikačního kódu.

V rámci této práce byly vytvořené Unit testy pro testování knihovny určené ke komunikaci s laserovým skenerem, a hlavně testy testující metody vyhledávacího algoritmu. Jako testovací

Framework byl použit NUnit. Jedná se o open-source Framework vycházející z JUnit, který je určený pro testování v Javě. NUnit poskytuje spoustu porovnávacích metod, které pocházejí ze třídy „Assert“. Mezi nejdůležitější metody této třídy patří:

- AreEqual - tato metoda je nejčastěji používána, a funguje tak, že porovnává rovnost dvou objektů a to očekávaný a aktuální.
- AreNotEqual - protiklad předchozí metody, porovnává jestli dva objekty nejsou rovné.
- AreSame - porovnává, jestli dva objekty mají stejnou totožnost.
- AreNotSame - porovnává, jestli dva objekty nemají stejnou totožnost.
- IsTrue - tato metoda kontroluje, zda je daná podmínka pravdivá.
- IsFalse - testuje, jestli je zadaná podmínka nepravdivá.
- IsNull - zjistí jestli je daný objekt roven nule.
- IsNotNull - ověří jestli zadaný objekt není roven nule.

Během vytváření testů byl dále použit „mockovací“ Framework „Moq“ a pro spouštění testů test runner, který je již integrován ve Visual studiu. Pro každou testovanou třídu byla vytvořena testovací třída, v které se nacházely testovací metody, testující každá jednotlivé funkce a metody z testované třídy. Na počátku každého testu se před provedením vytvořil konstruktor třídy a po ukončení jednoho testu se konstruktor vymazal. Díky tomu se zajistila nezávislost jednotlivých testů mezi sebou. V NUnitu se používají pro tyto funkce tzv. anotace a to konkrétně „SetUp“ pro nastavení všech potřebných parametrů před provedením testu a „TearDown“ pro smazání všech potřebných údajů po provedení jednotlivého testu. Anotace značící testovací třídu je „TestFixture“ a pro jednotlivé testovací metody se používá anotace „Test“. Díky těmto anotacím se pak dá přehledně spouštět jednotlivé testy pro jednotlivé třídy a metody a také lépe se hledají chyby. Jak vypadá jednoduchý test vytvořený v této práci je zobrazeno na následujícím kódu.

```
[TestFixture]
public class DetectAreaTests
{
   Mock<DetectArea> detectArea;
   [SetUp]
   public void TestSetup () {
      detectArea = new Mock<DetectArea>("DetectZone", new Points(-300, 2400),
          1500, 650);
   }
   [TearDown]
```

```
    public void TearDown () {
        detectArea = null;
    }
    [Test]
    public void InAreaTest () {
        Assert.AreEqual(true, detectArea.Object.InArea(new Points(-297.46,
            2322)));
        Assert.AreEqual(true, detectArea.Object.InArea(new Points(-278.13,
            2332.47)));
        Assert.AreEqual(true, detectArea.Object.InArea(new Points(-272.3,
            2318.06)));
    }
}
```

Výpis 10: Testování metody na zjištění jestli se bod nachází v detekční zóně

Jak je patrné z přiloženého kódu, tak se nejdříve nastaví správně anotace, jak již bylo popsáno výše a nastaví se, aby před vykonáním každé testovací jednotky vytvořil konstruktor třídy reprezentující detekční oblast, který je „mockován“. Dále bylo nastaveno vynulování instance třídy po ukončení každého testu. Tyto dvě metody se vytvoří automaticky před spuštěním každého testu v této testovací třídě. Nakonec byl vytvořený samotný test, který ve třech funkcích testuje tři různé body, jestli se nachází v detekční oblasti.

Na tomto principu bylo vytvořené testování nejdůležitějších součásti projektu, a to zejména knihoven obsluhující nějaké důležité funkce.

9.3 Testování a nasazení aplikace v provozu

Po testovacím období v kancelářských podmínkách a po vytvoření základních jednotkových testů, se následně přistoupilo k ostrému testování v provozu. Veškerý vytvořený software se nainstaloval na automatizační panel, jenž byl řídícím panelem konstrukce s manipulátorem a se skenerem. Nejdříve bylo nutné nastavit a otestovat komunikaci se skenerem a z OPC serverem. Dalším krokem bylo nastavení samotného skenovacího procesu. Jelikož skenování a následný sběr přepravních nádob manipulátorem probíhal ve dvou polohách. Celá konstrukce je rozdělená na levou a pravou část, kde se do každé části dá zasunout paletu nebo vozík z až třemi sloupci za sebou uložených přepravek, s tím že v každém sloupci bylo až osm přepravek na sobě. Díky tomu bylo potřeba pro obě strany konstrukce nastavit jiné parametry, a to zejména rozměry detekční zóny, z které algoritmus zpracovával dále data.

Jednotlivé palety s přepravními nádobami při naložení z obou stran, jsou vždy oddělené spodním dílem konstrukce, což zajišťuje, že při správném nastavení detekční oblasti, bylo dosaženo, že při skenovacím procesu se zpracovávaly data pouze z prostoru, kde se nacházela skenovaná paleta. Skener umístěný na manipulátoru skenoval z každé strany vždy z přibližně

stejných souřadnic, avšak aby bylo vyhodnocování co nejpřesnější, tak se do aplikace posílala na začátku každého skenovacího procesu souřadnice skeneru v ose X. Dalším problémem, jenž bylo třeba vyřešit během testování, bylo korektní nastavení posílání impulsů z krokového motorku do skeneru. Po několika pokusech se zdařilo nastavit posun z krokového motorku na enkodéru na 0,5 milimetrů, čímž se dosáhla velmi velká přesnost skenování.

Po provedení všech počátečních nastavení se dále přistoupilo k samotnému testování skenování přepravních nádob. Do obou skenovacích oblastí se naložily palety s přepravkami a začalo se skenovat obě oblasti. Samotné testování probíhalo tak, že se nejdříve provedl skenovací proces, kdy se na konci tohoto procesu vyhodnotily naskenovaná data algoritmem a následně se zapsaly data do příslušných OPC položek. Všechny tyto úkony ohledně skenování a vyhodnocování prováděla Windows služba, jenž již byla podrobně popsána v předchozích kapitolách.

Po každém skenovacím procesu a zapsání dat do OPC se provedla kontrola správnosti. Ta se prováděla pomocí ručního přeměření skenovaných přepravek, a hlavně pomocí vytvořené WPF aplikace sloužící k prohlížení jednotlivých skenů. Pomocí této aplikace se dalo podrobně zkontrolovat účinnost vyhledávacího algoritmu a postupně ho upravovat. Díky této aplikaci byly úpravy algoritmu velmi jednoduché, jelikož se na základě dat uložených z několika skenovacích procesů dalo touto aplikací algoritmus vždy bezpečně ověřit. Po každé změně programu se také spustily příslušné unit testy, aby se zamezilo vytvoření nějaké neočekávané chyby. Na základě testování byly unit testy dále rozšiřovány o nové testy, které testovaly nově opravené chyby.

Tímto způsobem se postupovalo několik hodin, než se vyzkoušelo všechny možné situace, které mohly během skenovacího procesu nastat. Během vykonávaní měření se podrobně sledovaly také logy, jež byly službou zapisovaný. Na základě těchto logů byla aplikace dále vylepšována a opravována. Dalším krokem v testovacím procesu bylo testování již samotného nastavení navádění manipulátoru na výsledné souřadnice a přenos přepravní nádoby na dopravníkový pás. Tento úkon měla na starosti již automatizační část, kterou se tato práce nezabývá, jelikož nebyla součástí našeho úkolu, avšak ji zde zmiňuji, protože byla velmi důležitá k nastavení vyhledávacího algoritmu, tak aby byl manipulátor naveden co nejpřesněji. Během této testovací části se ještě navádění manipulátoru na výsledné souřadnice podrobně sledovalo a hlídalo se případné chyby.

Po několika dnech testovacího provozu této aplikace a odstranění většiny nedostatků se mohlo přistoupit k testování v ostrém provozu. Tato testovací část trvala 48 hodin a nesmělo zde dojít k závažným chybám během skenování a vyhodnocení. Celý skenovací proces a následný úkon de-paletizace již byl řízen zaškoleným personálem obsluhující tuto část výroby. Na aplikacích v tomto režimu se nesmělo nic měnit, až na změny v konfiguračních souborech. Jinak vše fungovalo bez většího zásahu a sledování. Po úspěšném ukončení této testovací části již byla naše práce ukončena a proběhlo předání a zaškolení personálu a objednavatele na obou vytvořených aplikacích.

10 Tvorba dokumentace

Posledním úkolem v této práci bylo vytvoření uživatelské dokumentace a zaškolení personálu, jak již bylo popsáno v předchozí kapitole. K oběma vytvořeným aplikacím byl vytvořený podrobný manuál, který se skládal z těchto kapitol:

- Seznam vytvořeného softwaru
- Umístěni softwaru
- Instalace softwaru potřebného pro běh aplikací
- Instalace a aktualizace aplikací
- Podrobný popis konfiguračních souborů a jejich položek
- Stručný popis funkce aplikací

Celý uživatelský manuál vytvořený k tomuto projektu je umístěn v příloze této práce. V první kapitole je seznam vytvořených aplikací se stručným popisem jejich účelu. V další kapitole je popsáno umístění obou aplikací, pro nalezení konfiguračních souborů. Ve třetí kapitole manuálu je popis aplikací, které je nutné nainstalovat pro správný běh aplikací. Jedná se o Microsoft .Net Framework ve verzi 4.6.1 a OPC Core Components potřebných ke správnému chodu OPC komunikace.

V další kapitole je manuál zaměřený na popis instalace obou vytvořených aplikací. Instalace probíhá pomocí instalačních souboru, které po spuštění a vybrání místa instalace se automaticky samé nainstalují. Dále je zde popsán postup, jak se provádí automatická aktualizace. Princip aktualizace spočívá vtom, že se nová verze aplikace nahraje na server do příslušné složky a odkaz na tuto složku je nastavený v konfiguračním souboru. Aby se aplikace aktualizovala je nutné ji znovu spustit, což v případě Windows služby vyžaduje restart uživatelem, jelikož je v nepřetržitém provozu. Po restartu, aplikace zjistí, že je na serveru nahrána nová verze a automaticky si stáhne nové verze souboru a restartuje se.

V páté kapitole manuálu jsou podrobně popsány všechny konfigurační položky, a to jak jejich funkce, tak případné jednotky, v kterých jsou nastavené. V poslední kapitole je stručně popsána funkce vytvořených aplikací.

Na základě této uživatelské dokumentace, proběhlo zaškolení personálu obsluhující část výroby, kde je aplikace umístěna a předán vedení zadavatelské firmy.

11 Závěr

Cílem této diplomové práce bylo navrhnout a vytvořit Windows službu, která obsluhuje laserový skener, komunikuje s PLC panelem pomocí OPC technologie, a hlavně zpracovává a vyhodnocuje naskenovaná data. Byl tedy vytvořen vyhodnocovací algoritmus, jenž má za úkol nalézt přepravní nádoby z naskenovaných dat a výsledky tohoto algoritmu se následně zapíší do OPC.

Pro kontrolu naskenovaných dat a zobrazení jich pomocí 3D modelu byla vytvořená WPF aplikace. Součástí vytvořeného softwaru jsou také unit testy, testující funkčnost knihoven vytvořených v rámci této práce.

Po vyhodnocení požadavků a vytvoření analýzy a následného návrhu aplikací, byly vytvořené a nasazené do reálného provozu obě popisované aplikace.

Vyhodnocovací algoritmus byl přizpůsoben vzhledem na efektivitu nalezení přepravních nádob, které se mohly nacházet v místě nasazení. Při dalším nasazení těchto aplikací na jiné místo a pro jiný druh přepravních nádob by bylo potřeba upravit nebo změnit vyhodnocovací algoritmus za jiný.

Aplikace byly vytvořené tak, aby byly schopné zareagovat na jakoukoliv neočekávanou situaci, a v případě selhání vyhodnocování, nebo nastání chyby během skenovacího procesu, byly schopné opět provést následující skenovací proces.

Z důvodu pohodlné instalace byly pro obě aplikace vytvořené instalátory, umožňující jednoduchou instalaci. Pro zjištění nové verze aplikace a aktualizování na ní, byla do obou aplikací přidána funkce automatické aktualizace.

V rámci této práce proběhl vývoj od analýzy a návrhu po nasazení do provozu bez větších problému, a obě vytvořené aplikace jsou nasazené v reálném provozu. Z toho vyplývá, že všechny požadované a naplánované úkoly a funkce byly v rámci této práce splněny.

V budoucnu, by se měla tato aplikace dále rozšiřovat do jiných části provozu v továrně kde byla nasazena, a proto bude potřeba vylepšit některé části, jako je změna vyhodnocovacího algoritmu, podle toho, v jaké části provozu se aplikace nachází. Dalším možným vylepšení by bylo zapracovat na zefektivnění aplikace pro zobrazení 3D modelu a to z toho důvodu, že zobrazení naskenovaných dat vzhledem k jejich množství není úplně svižné a optimální.

Aplikace vytvořené v této práci slouží pro zefektivnění výroby dílu do automobilových převodovek a usnadňují zaměstnancům práci s překládáním těžkých přepravních nádob z těmito díly.

Literatura

[1] Úvod do softwarového inženýrství [online]. ©2002[cit. 06.04.2018]. Dostupné z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf

[2] Windows Presentation Foundation – Wikipedie [online]. 2017[cit. 06.04.2018]. Dostupné z: https://cs.wikipedia.org/wiki/Windows_Presentation_Foundation

[3] MVVM: Model-View-ViewModel. Největší český web zaměřený na .NET framework [online]. ©2018 [cit. 06.04.2018]. Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[4] Obrázek návrhového vzoru MVVM [online]. [cit. 06.04.2018]. Dostupné z: https://dotnetportal.blob.core.windows.net/files/Windows-Live-Writer/MVVM-Model-View-ViewModel_DB59/demo_2.png

[5] Lidar – Wikipedie [online]. 2018[cit. 07.04.2018]. Dostupné z: https://cs.wikipedia.org/wiki/Lidar

[6] Lidar - Wikipedia [online]. 2018[cit. 07.04.2018]. Dostupné z: https://en.wikipedia.org/wiki/Lidar

[7] 2D-LiDAR senzory Lms4xx [online]. ©2018 SICK AG[cit. 07.04.2018]. Dostupné z: https://www.sick.com/cz/cs/reseni-mericich-a-detekcnich-aplikaci/2d-lidar-senzory/lms4xx/lms400-2000/p/p112350

[8] Odrazivost materiálů [online]. [cit. 07.04.2018]. Dostupné z: http://engineeronadisk.com/book_plcs/images/plc_disc_sens35.gif

[9] Obrázek pracovní oblasti laserového skeneru Sick Lms 400 [online] [cit. 08.04.2018]. Dostupné z: https://www.sick.com/media/ZOOM/2/12/212/IM0039212.png

[10] Katalogový list s operačními instrukcemi pro Sick Lms 400 [online]. ©2013-2017[cit. 08.04.2018]. Dostupné z: https://sick-virginia.data.continum.net/media/docs/8/98/698/Operating_instructions_LMS400_Laser_measurem

[11] Programování Windows Services. Největší český web zaměřený na .NET framework [online]. ©2018 [cit. 13.04.2018]. Dostupné z: https://www.dotnetportal.cz/clanek/194/Programovani-Windows-Services

[12] Log – Wikipedie [online]. 2016[cit. 13.04.2018]. Dostupné z: https://cs.wikipedia.org/wiki/Log

[13] OLE for Process Control – Wikipedie [online]. 2015[cit. 14.04.2018].Dostupné z: https://cs.wikipedia.org/wiki/OLE_for_Process_Control

[14] Popis OPC serveru [online]. ©2011[cit. 14.04.2018]. Dostupné z: http://www1.fs.cvut.cz/cz/u12110/site/Nosek-OPC_server.pdf

[15] OPC Data Access - Wikipedia [online]. 2017[cit. 15.04.2018]. Dostupné z: https://en.wikipedia.org/wiki/OPC_Data_Access

[16] 1. díl - Úvod do testování softwaru v C# .NET [online]. ©2018 itnetwork.cz. Veškerý obsah webu [cit. 15.04.2018]. Dostupné z: https://www.itnetwork.cz/csharp/testovani/uvod-do-testovani-softwaru-v-csharp-net

[17] Jak začít psát unit testy - Miroslav Holec. Miroslav Holec - Software Architect & Evangelist [online]. 2016[cit. 15.04.2018]. Dostupné z: https://www.miroslavholec.cz/blog/jak-zacit-psat-unit-testy

12 Přílohy

I. Příloha na CD

Příloha obsahuje složku RtsDyAut, v které se nachází zdrojový kód všech vytvořených aplikací. Příloha dále obsahuje uživatelskou dokumentaci.